AFRL-HE-WP-TR-2006-0031

AIR FORCE RESEARCH LABORATORY

Tools for User-assisted Behavioral Monitoring of Distributed Information Networks

Kaliappa Ravindran

Department of Computer Science
The City College of CUNY
Convent Avenue at 138th Street
New York, NY 10031

June 2005

Final Report for November 2003 to June 2005

Human Effectiveness Directorate
Warfighter Readiness Research Division
Logistics Readiness Branch
2698 G Street
Wright-Patterson AFB OH 45433-7604

TECHNICAL REVIEW AND APPROVAL

AFRL-HE-WP-TR-2006-0031

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

//SIGNED//

JEFFERY C. WHARTON, LtCol, USAF
Deputy Chief, Warfighter Readiness Research Division
Human Effectiveness Directorate

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* June 2005	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* November 2003 - June 2005
4. TITLE AND SUBTITLE Tools for User-assisted Behavioral Monitoring of Distributed Information Networks		**5a. CONTRACT NUMBER** F33615-03-1-6385
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER 62202F
6. AUTHOR(S) Kaliappa Ravindran		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER 1710D219
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Department of Computer Science, The City College of CUNY, Convent Avenue at 138th Street, New York, NY 10031		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Materiel Command, Air Force Research Laboratory, Human Effectiveness Directorate, Warfighter Readiness Research Division, Logistics Readiness Branch, Wright-Patterson AFB OH 45433-7604		**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/HEAL
		11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-HE-WP-TR-2006-0031

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

Cleared as AFRL/WS-06-1525, 15 Jun 06.

14. ABSTRACT

This research focused on providing an event specification approach to enable a flexible monitoring and control of distributed real-time information systems (DIS). The research constructed an object-oriented Application Programming Interface (API) that allows the monitoring of compliance to properties deemed as critical for function of a DIS. Users can prescribe critical properties in the form of *event predicates*, which are Boolean conditions on the externally visible computation state among the DIS nodes. Events depicting the violation of critical properties, for instance, due to information attacks and/or failures, can be detected by checking if event predicates have become true.

15. SUBJECT TERMS

Distributed Real-Time Information Systems (DIS), Application Programming Interface (API)

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Edward S. Boyle
a. REPORT UNCLASSIFIED	**b. ABSTRACT** UNCLASSIFIED	**c. THIS PAGE** UNCLASSIFIED	SAR	66	**19b. TELEPHONE NUMBER** *(include area code)*

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. 239.18

THIS PAGE LEFT INTENTIONALLY BLANK

Contents

PROJECT SUMMARY

The project focused on providing an event specification approach to enable a flexible monitoring and control of distributed real-time information systems (DIS). The project constructed an object-oriented API ('Application Programming Interface') that allows the monitoring of compliance to properties deemed as critical for functioning of a DIS. Users can prescribe critical properties in the form of *event predicates*, which are boolean conditions on the externally visible computation state among the DIS nodes. Events depicting the violation of critical properties, say, due to information attacks and/or failures, can be detected by checking if event predicates have become true.

A general paradigm that underlies in our approaches is a DETECT $\longrightarrow$ REACT $\longrightarrow$ REPAIR cycle executed at computational time-scales which are much faster than the domain-specific problem time-scales. In a way, the system consciously allows a limited (and controlled) amount of damage at the system periphery but orchestrates corrections before these damages percolate into deeper layers of the system. This is unlike the PREVENT paradigm which underlies many of the existing approaches that strive to prevent any damage at all (so they do not scale well as systems become more complex, diverse, and heterogeneous).

Our monitoring approach manifests in two facets:

1. Designing a specification meta-language which allows the interface behavior of a system to be prescribed through possible occurrences of events in the environment;

2. Developing application case studies that employ the interface specification approach to structure their control and recovery activities.

Application-level users play an active role in prescribing the event conditions and the underlying domain-specific event formulations. The goal is to increase the effectiveness of the decision-making capability of users in sifting through various complex and real-time scenarios occurring in the problem-domain.

1 Project Goal: Distributed monitoring of information systems

We begin with generalized structure of a distributed information system (DIS) to expose the underlying problems, and then narrow down to the programming models we have developed for system-level monitoring as building blocks of solutions.

1.1 Behavioral view of distributed information systems

A real-time DIS for military applications interacts with an external environment that consists of:

- Data sources, namely, sensors of physical environment variables;
- Data sinks that map computation results into physical environment actuators.

The DIS consists of multiple computation nodes (or sub-systems) that carry out a battlefield management task, based on the information fed in from the environment. See Figure 1. Examples of such information networks include networked-sensor systems for terrain surveillance in 'digital battlefield' settings and multimedia-based collaborative information dissemination systems[1].

Human commanders and/or intelligent agents at distinct geographic locations are part of the DIS, interacting with one or more nodes to obtain strategic decision support. In a 'digital battle field' for instance, the environment may consist of infrared sensors to collect data about enemy troop and machinery movement in a certain terrain. The computation subsystems are interconnected by an information distribution network that make the relevant pieces of data available to these subsystems at various points in time, to enable them carry out the task.

1.2 Ontological view of the model of adaptive DIS

A general paradigm that underlies in our approaches is a DETECT $\longrightarrow$ REACT $\longrightarrow$ REPAIR cycle executed at machine-level time-scales which are much faster than the application-perceivable time-scales. In a way, the system consciously allows a limited (and controlled) amount of damage but orchestrates corrections before these damages escalate into seriously higher level proportions. This is unlike the PREVENT paradigm which underlies the "cryptographic tools" based protection methods in many existing approaches. See Figure 2.

As systems become more and more complex (due to size, scale, heterogeneity, and diversity), a prevalent thinking among system designers is that it may be quite difficult to provide a 100% security and assurance, and it is rather prudent to learn to live with some limited damages to

[1]The evolution of new types of information networks has become possible mainly because of the low cost availability of workstations and personal computers with high CPU speeds, high bandwidth networks to interconnect computers, and functionally rich I/O devices with local processing, miniaturization and/or mobility support.

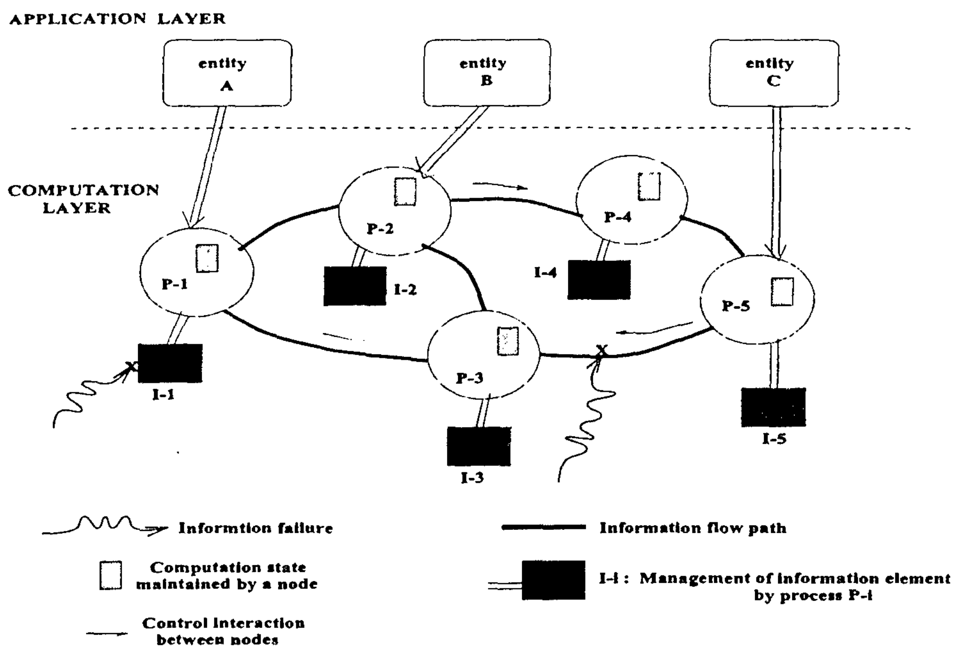


Figure 1: A bird's view of a distributed information system (DIS)

the system operations. For instance, even if we have a 99.9999 ⋯ % secure cryptographic tool for data transmission between system components, an attacker can cause damage at the level of 'information' disseminated by end-users of the system from an otherwise securely transmitted 'data' through the system components !! For example, a sensor may simply corrupt the data collected from an external environment, and then transmit this corrupted data to the end-user over a highly secure communication channel. The end-user operating on such a corrupted data may then inadvertently induce incorrect behavior at the application level. Thus, rather than trying to secure everything at the data level, it may be more meaningful (and practical) to install watch dogs and recovery mechanisms at appropriate points of information dissemination in a DIS. The project we have completed reflects this paradigm shift in system designs.

1.3 Monitoring of critical properties

The acceptability of a system may be based on whether its observed behavior meets certain criteria that are absolutely essential in order for it to function without interruption. A prescription of these essential criteria may be deemed as a *critical property* of the system. A critical property may be associated with specific *symptoms* that can be observed by a management module.

In a broad sense, critical properties may cover many aspects of system behaviors as captured by various symptoms, such as those given below:

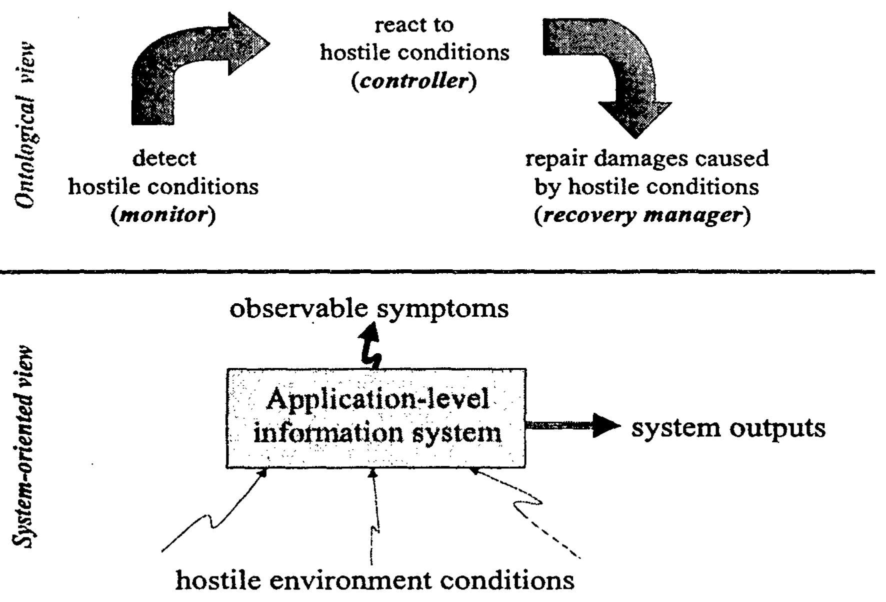

Ontological view
react to hostile conditions (*controller*)
detect hostile conditions (*monitor*)
repair damages caused by hostile conditions (*recovery manager*)
System-oriented view
observable symptoms
Application-level information system
system outputs
hostile environment conditions

Figure 2: DETECT $\longrightarrow$ REACT $\longrightarrow$ REPAIR paradigm in our approach

- 'quality of service' offering to the application, measurable as macroscopic performance indices (e.g., the probability of image presentation glitches due to network delays exceeding a given limit);
- Faults occurring in system components, whose effects can propagate into other parts of the system (e.g., the crash of a replica node that reduces the 'data resiliency' in a 'highly available data' system);
- Topology changes in subsystem interconnections, arising from the addition and/or removal of subsystems (e.g., inducting a new node in the network that may increase the delay along a data path).

It is up to the application to determine what constitutes a mandatory property that must be met during a computation.

Regardless of the criteria in enunciating a critical system property, it is necessary to continually monitor a system behavior, so that reasoning about when the system meets its critical expectations and when it fails to meet can be made. Due to the real-time nature of information, any detection of critical property violation is also time-sensitive.

1.4 Relationship to deep packet filtering

Our approach is based on using *proxy agents* to implement a monitor for critical property violations in a DIS. The project may be viewed as related to the area 'deep packet filtering' where the management agents in a DIS validate the 'data contents' of a message exchanged between the system nodes against a set of rules supplied by the application[2]. Our approach differs from 'deep packet filtering' in the following ways.

A comprehensive detection of application-level attacks is difficult, if not impossible. Because, it requires a large rule space that may either not be expressible in concise forms or require traversal of a large state space before validating a particular client action (as carried in application-level packets). For scalability and operational reasons, the monitor-based DETECT $\longrightarrow$ REACT $\longrightarrow$ REPAIR approach is the first level defense at the perimeter of a system. The idea is that if an attack does occur, it can be detected soon enough before a serious damage gets done to the system (i.e., before an attacker reaches the inner layers of the system).

1.5 User-level participation in monitoring

The prescription of critical properties may sometimes involve many state variables, and require capturing the causal and timing relationship between them. In a QoS-oriented web application for instance, the page access delays, page access failure rate, and available server bandwidth are some of the parameters to be considered in determining if a denial-of-service attack is taking place on the server. Their relationship may often not be expressible in a closed form, due to the non-separable ways in which one parameter may impact the others. In general, only imprecise state information may be available in the system, compounded by the lack of quantitative relationship between the state variables.

The implementation of event predicates under imprecise and non-exact handles on the state conditions requires users to actively participate in the event monitoring process. Application-level participation requires an ability to reconfigure event prescriptions as the domain-specific computation progresses forward and to notify users when prescribed events do occur at run-time.

The development of monitor tools in our approach involves two project phrases: i) a programming model that unifies the notion of time-scales into the event conditions (i.e., predicates) for detection purposes; and ii) incorporating our monitor approach in application-level case studies. The project phrases (i) and (ii) have been completed, as indicated by the publication of papers listed as #1-#5 in this report — see section 7.

[2]The semantics-oriented packet validation that underlies 'deep packet filtering' is quite different from the existing notions of SNMP-style packet filtering which is mainly based on a small set of rules pertaining to the communication sub-system operations (e.g., number of packets from an IP address exceeding a certain limit).

2 Limitations in current event specification models

Currently available specification models for 'classical' real-time control (as in avionics and industrial applications) often employ an *integrated* approach, whereby the monitoring and control functionality are built into the computation modules that constitute the target system being controlled. The monolithic structure of an 'integrated' programming view makes it difficult to re-use the functionality built into one system for deployment in other systems and to incorporate user-level participation. This limitation arises from two aspects:

- The sensor and actuator data generated in various target systems may impose different computational requirements (such as representation and user-level manipulations — e.g., MPEG-based image compression);
- Different target systems often exhibit dissimilar temporal behaviors (such as the control actions being subject to hard or soft deadlines, and requiring different time-scales of responses).

The difficulties in a re-use of system designs manifests as redundancies in distributed software development. This can in turn slow down the evolution of newer types of information networks, besides increasing the cost of development and maintenance of distributed software.

. Thus, we need a *modular* approach whereby the temporal behavior and the computational view of the target system can be captured through generic event specification modules through a GUI, instantiated with problem-specific parameters. Our project theme is based on this approach.

3 Programming models for real-time event detection

Our monitor API incorporates tools for prescribing events in terms of a meta-language, whereupon the event prescriptions can drive a distributed algorithm to monitor these events. In general, a critical property may be represented by a condition on the state variables S exported through a programming interface that is visible to application entities (i.e., users). An associated system event is said to occur when user-prescribed condition on the problem-specific state holds, i.e., a predicate $L(\Psi(S)) =$ true where $L(\cdots)$ is a logical formula and $\Psi(\cdots)$ denotes a state mapping for use by the applications. When the event occurs, it is understood that the system has indeed failed to meet the corresponding critical requirement.

The condition $L(\Psi(S))$ is supplied by a user to his/her local computation node. Once an event predicate is prescribed, a distributed 'event detection' algorithm is executed by the monitoring agents to evaluate the predicate at various time points using the instances of interface state $\Psi(S)$ maintained by the computation nodes. The event is deemed to have occurred when the predicate evaluates to true, whereupon, this result should be propagated to the computation nodes to initiate a recovery.

We view our approach as *event filtering*, wherein an event filter is a programmable selection criterion for classifying or selecting events from a stream of events. The event filter is basically prescribed in a high-level declarative language that is compiled into the event monitor portion of the programming system. At the programming level, event observations predicated by boolean conditions on the state space of problem domain reduces the number message exchanges, and hence the amount of state tracking required by the monitoring nodes.

We have studied the programming model in the application-domains of QoS management for 'content distribution networks' and video transport networks. The papers #2 and #4 provide the details of how our programming model is conceptualized from an event specification standpoint (to enable a system-level implementation).

4 Proxy-based protocol-level control of network operations

In a particular instantiation of our model, light-weight proxy-agents maintain simple rule sets that allow the detection of deviations from a prescribed system-level behavior — which is basically a critical property of the system. Upon detecting such critical property violations, more heavy-weight proxy agents with more elaborate rule sets replace the erstwhile agents and prevent the damage from escalating into deeper layers of the system.

In a multimedia presentation system for example, 'data slips' may occur at the user-level due to system and network delays on 'data messages'. Keeping this 'data slip' within tolerance limits is a critical requirement to maintain the cohesiveness of multimedia presentations. When message delays are small — which is a normal case, simple rules prescribing the delivery of 'data messages' as and when they arrive suffice without causing data slips. However, when message delays are high and random, complex rules are required to maintain the synchronization between various messages: such as the voice annotation to a click-and-point operation on a shared multimedia window should maintain the right amount of time interval and ordering between the corresponding 'data messages'. In our model, the monitor consists of communication agents in the system which detect violations of the media-level synchronization requirements by observing the 'data slips' over network-level time-scales. A recovery basically involves switching from the simple rule set (i.e., light-weight agents) to a complex rule set (i.e., heavy-weight agents).

We have studied this idea in the application-domains of QoS management for 'content distribution networks' and video transport networks. The papers #1, #3, and #5 provide the details of how the idea is implemented.

5 Advantages of our monitor-based approach

We broaden the classification of events through user-specified boolean conditions exercisable on the externally visible system state. Conditions that evaluate to preset values (i.e., true or false) can be treated as indicative of deviations from expected critical behaviors. Thus our approach provides a uniform framework to capture application-specific semantics of 'system properties' and 'events'. This aspect is borne from the prescription of user-profilable events that can parameterize a general-purpose event detection algorithm.

In rate-based congestion control of networks as case study (see paper #1), the protocol mechanisms for detecting congestion in network paths are separated from the policies for adjustment of data sending rates. The former functionality is then instantiated into our general-purpose monitor module by prescribing the range of network parameter values that may signal 'congestion events'. The rate adjustment functions are implementable by the data sources.

As can be seen, our monitor-based view of a distributed application allows the separation of anomaly detections from recovery functions. This leads to the ease of prototyping of various policy mechanisms for monitoring and control — as advocated in 'policy-driven network' structures. Our modular approach thus eliminates redundancy and duplication in software development for large scale complex systems.

6 Project conclusions

Application-level monitoring of distributed real-time information networks to facilitate the detection of anomalous and faulty situations is the focus of our project. We support this monitoring by employing an object-oriented approach to characterizing system behaviors, and determining the monitorable events therefrom. The main innovative elements of our approach are as follows:

- Integration of the 'flow of real-time' into program-level notion of 'sensors' and 'actuators'

  This involves the decomposition of a DIS into canonical modules: 'sensors' that translate the state changes occurring in a problem-domain into monitorable events, and 'actuators' that parameterize recovery actions with problem-specific control signals. Thus 'sensors' and 'actuators' are object-oriented abstractions of physical devices and/or logical devices embedded in a real-time DIS.

- Meta-level specification of events in terms of generic operators and applicative functions

  This provides the ability for users to prescribe what events are of interest to them (i.e., the events to be monitored). Since events can be prescribed at various granularity levels of the problem-domain state space, the monitor can be programmed to monitor different types of events in the same application or customized to suit different applications.

- Application-level case studies of our event-oriented programming model

  Our model has been validated by implementing adaptive network applications with functional elements and tools that characterize the model. The case studies are 'content distribution networks' and video multicast transport. These applications which are inherently complex in nature have been implemented with relative ease and modularity by virtue of using our model.

7 List of publications from project

1. K. Ravindran and X. Liu. *Service-level Management of Adaptive Distributed Network Applications*, M. Malek et al. (Eds.): ISAS 2004, LNCS 3335, Springer-Verlag Berlin, Heidelberg, 2005.

2. K. Ravindran. *Programming Models for Behavioral Monitoring of Distributed Networks*, Proceedings of 38th Annual Hawaii International Conference on System Sciences, Big Island (HI), January 2005.

3. K. Ravindran and Jun Wu. *Performance Management Protocols for Adaptive 'content distribution networks'*, Proceedings of IEEE International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA 2005), Orlando (FL), June 2005.

4. K. Ravindran and Jun Wu. *Event-based Programming Models for Monitoring of Distributed Information Systems*, Proceedings of 9th IEEE International Symposium on Distributed Simulation and Real-time Applications (DS-RT 2005), Montreal (Canada), pp.236-245, October 2005.

5. K. Ravindran and Jun Wu. *'dynamic protocol plug-in': a Middleware Provision for Enhancing Network Service Performance*, To appear in proceedings of 1st International Conference on Communication System Software and Middleware (COMSWARE 2006), New Delhi (India), January 2006.

Appendix 1

Service-level Management of Adaptive Distributed Network Applications[‡]

K. Ravindran[1] and Xiliang Liu[2]

[1] City College of CUNY and Graduate Center,
Department of Computer Science,
Convent Avenue at 138th Street,
New York, NY 10031, USA
ravi@cs.ccny.cuny.edu

[2] CUNY Graduate Center, Computer Science,
365 Fifth Avenue,
New York, NY 10016, USA
xliu@gc.cuny.edu

Abstract. The paper is on generic service-level management tools that enable the reconfiguration of a distributed network application whenever there are resource-level changes or failures in the underlying network subsystems. A network service is provided to client applications through a protocol module, with the latter exercising network infrastructure resources to meet the client requirements. Client requests for a network service instantiate the underlying protocol module with parameters specified at the service interface, along with a prescription of service quality to be enforced therein. At run-time, a management module may automatically monitor the service compliance to client-prescribed requirements, and notify the client whenever a service degradation is detected. The paper proposes a 'function'-based model of service provisioning to realize our management approach. In this model, a service prescription conforms to generic interface templates based on an enumeration of the service attributes visible to clients, and how these attributes logically relate to one another in composing a client-level quality expectation. Our management model is independent of the specifics of problem-domain, which simplifies the development of distributed adaptive applications through a 'software reuse' of the management module. The paper demonstrates the usefulness of our model with a case study of CDNs.

1 Introduction

Network-centric distributed applications are evolving in the form of requiring far diverse and widely varying service capabilities from the network infrastructure (such as electronic banking, tele-medicine, and tele-shopping). Network infrastructures are also evolving to augment their capabilities in terms of diverse

[‡]Part of the work on signaling interface for network monitoring is currently supported by AFRL-HE grant: F33615-03-1-6385 (2003). Part of the work on case study generation and simulation is supported by PSC-CUNY Award: 66441-00-35 (2004).

criteria and to offer these capabilities in a form usable by applications [1]. To balance application evolutions on one hand and infrastructure evolutions on the other without constraining either, comprehensive network management tools and paradigms are required that can allow applications to interwork with network infrastructures in a flexible and extensible manner. A part of these challenges arises due to infrastructure outages and/or changes that may occur dynamically during an application execution (e.g., increase in the access latency on a web page due to the crash of a 'content distribution' agent node). Our paper identifies a suitable *network service model* to support the development of applications that can adapt to the capabilities offered by network infrastructures.

The current management standards, TINA and DCOM [2,3], advocate the partitioning of a distributed network application into two types of entities: *service provider* (SP) and *service user* (or client). The SP maintains a repertoire of protocol mechanisms that are capable of offering network services to clients. For service delivery, a protocol mechanism exercises the infrastructure *resource* components placed at different sites of the network. A client application controls the extent to which network infrastructure resources are exercised, based on its service-level requirements. Our work on network service models purports to 'liaison' these two aspects of network application development. See Figure 1.

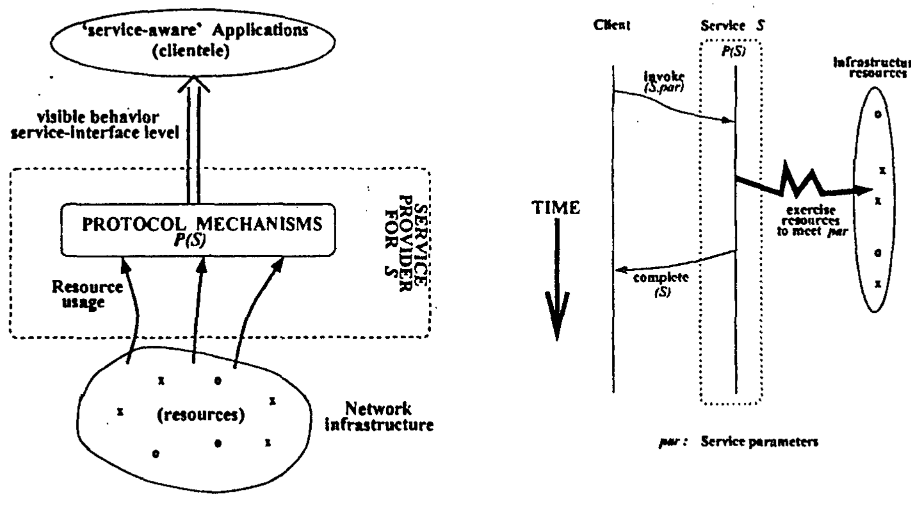


Fig. 1. Layers of functions in a service provisioning system

A protocol $P(S)$ exercises the infrastructure resources in delivering a service S to clientele. The internal state maintained by $P(S)$, which abstracts the resource usage, may be mapped onto service-level parameters visible to clients. A client may determine the feasible service behavior therefrom, to compare with the service obligations expected of S. Any deviation in the actual service behavior from the expectations is symptomatic of resource-level failures (e.g., excessive packet loss seen by end-users indicates a heavy bandwidth congestion along the network path). Thus, *behavioral monitoring* of network services is a necessary element of client-level reconfiguration mechanisms.

From a service-level programming standpoint, the monitoring activity may be structured as generic functions that can be instantiated with problem-specific parameters. For instance, how often the state variables characterizing a service behavior are sampled is application-dependent, while the distributed algorithm to sample the state at various network nodes can itself be generic. Our model of network services allows developing the monitor and control functionalities in a *service-neutral* manner, making them usable across a wide range of applications. The service-neutrality of our model arises from its 'functional' orientation, where a service prescription conforms to a generic template based on enumeration of service features and how they logically relate to one another. A management module interworks with the service and client modules through a generic schema, for monitoring service compliance to client-prescribed quality requirements. The monitor is realizable by a set of software agents that can be implanted in the target client and service modules, and be projected onto the problem-specific parameter space to facilitate their interworking with the target modules. A re-use of the management agents across different applications may significantly simplify the development of distributed networking software.

In our model, service-level features are described by *attribute* parameters exported to clientele. A client application instantiates a network service, prescribing the required service quality in the form of logical relations on the attribute values. This prescription in turn allows the monitoring of service compliance to client requirements by the management module at run-time. Dynamic reconfiguration of clients may be triggered by notifications of service degradations from the management module. The paper describes case studies of network applications to bring out the usefulness of our management model.

The paper is organized as follows. Section 2 provides a management-oriented view of network service offerings. Section 3 describes our application-oriented function-based model of network management. Section 4 positions our model in the light of existing management paradigms. Section 5 provides a case study of the model on 'content distribution networks'. Section 6 concludes the paper.

2 Management-oriented view of service offerings

The functions of a network are made available to client applications through a service interface that exports well-defined service features and capabilities, i.e., *service attributes*. A client may exercise one or more features to obtain a desired quality of service delivery. For example, the 'accuracy level' of time information is a feature of Network Time Service (NTS) exercisable by a client-prescribed 'minimum accuracy' parameter — say, to time-stamp banking transactions. The protocol employed at network infrastructure level to provide a service (e.g., 'sync' message exchanges between NTS nodes) is hidden from clients.

2.1 Service behaviors visible to clients

How a service 'quality' as seen by clients is affected by the changes in infrastructure resources constitutes a service behavior. Given a client-prescribed quality expected of a service S, different protocol modules $P(S), P'(S), \cdots$ may exhibit distinct behavioral profiles, i.e., offer different levels of service S for a given amount of infrastructure resources.

From a client's perspective, two services are identical if their externally visible behaviors for a given a set of parameters are the same. As example, consider a 'circuit switched' service offered in POTS-style telephone switching networks [5]. A management-oriented view of 'circuit switched' service is one of a 'network link' that offers transfer delay with variance ≈ 0 around a mean d for each data unit (where $d = \frac{\text{size of data unit}}{\text{link bandwidth}}$). This view, while subsuming traditional POTS-style implementations that dedicates a 'copper wire' between end-points to realize voice 'links' (with implicit guarantees of bandwidth), also encompasses 'packet switching' implementations that exhibit a similar delay behavior — at least over an operating range of interest. In general, service differentiation may be in terms of measurable parameters of externally visible behaviors.

Changes in the parameters of a service offering depict macroscopic behavior indices (or symptoms) at the service interface, reflected onto from infrastructure changes. For example, a symptom of network congestion is the end-to-end data transfer delay over the network exceeding a limit [6, 7]. In general, a service behavior can be dynamic in nature, i.e., service features can change in the midst of service provisioning due to changes in the underlying infrastructure elements.

The service attributes exported by the SP may pertain to performance, availability, functionality, and the like. From a client standpoint, the quality expectation may be some combination of these attributes. With such service-aware clients, any degradation in the level of service offering should be within the tolerance limits of attributes prescribed by clients.

2.2 Programmability of service offerings

Clients invoke a network service S by prescribing a set of parameters. The SP instantiates a network protocol $P(S)$ by mapping the client prescriptions to protocol-level parameters [8]. The latter allow exercising the infrastructure resources by the functions that compose of $P(S)$. In general, a higher level of service obligation to clients requires an increased usage of resources; likewise, a reduced resource availability lowers the service quality. The parameterizable execution of $P(S)$ allows a macroscopic control of infrastructure resource usage to meet the client-prescribed service parameters. In the NTS example, the client-specified resolution and accuracy levels of time information influence the frequency of 'time sync' message exchanges between NTS nodes: higher accuracy requiring more frequent messages. The relationship between resource usage and service quality is embodied into the functions that realize $P(S)$.

Dynamically occurring infrastructure resource outages and/or failures (such as component upgrades or removals) may manifest as observable behavioral changes in a service offering. Based on a quantitative assessment of these changes, an *adaptive* client application may adjust its service parameters to match the available network resources: relax service expectations when resources are limited. In some cases, even with sufficient resources, a client may wish to relax its service expectations — for reasons such as usage-sensitive service tariffs.

Guaranteeing a minimal service obligation to clientele against severe failure conditions in the infrastructure depicts 'service availability'. If availability takes precedence over performance, the SP may employ an operationally safe protocol despite it being heavy-weight in terms of resource usage. Such a protocol incurs a bounded recovery time when failures occur. On the other hand, if performance is more important, a light-weight protocol may be employed — though it is unsafe. For example, a 'nack'-based protocol may be employed for a 'end-to-end data transfer' service when packet loss in the network is low, and an 'ack'-based protocol otherwise. The latter guarantees a correct 'data transfer' between users but at a lower transfer rate. In general, the client-level prescriptions of a service should capture the tradeoffs between availability and performance.

2.3 Time-scales of service monitoring

Consider the possibility of a client detecting service degradations through symptoms observed in the problem-domain. Due to slower dynamics of the problem-specific state analyzed by clients, the latency in such a detection may be higher. On the other hand, an automated detection of the symptoms of network service degradation involves simply comparing the client-prescriptions of expected service behaviors and the parameters of actual service offering from the network. Such a detection can be realized over a much faster time-scale. Figure 2 shows our experimental studies on the time-scales of congestion detection at various layers of a video distribution network. In these studies, the human-perceived quality degradation at the video receivers is on a slower time-scale, when compared to the receiver agent-level detection based on the fluctuations in observed frame loss rates. Accordingly, a recovery triggered by user-level notifications (say, through a GUI on receiver windows) incurs higher latency, and is less responsive to network congestions. In general, program-level symptoms can be detected much sooner than the corresponding problem-domain anamolies.

A service degradation can be detected by identifying specific patterns of state changes in a computational projection of the problem-domain. Accordingly, adaptive service provisioning can be realized by distributed algorithms that interpret the monitored program-level state variables at computational time-scales (such as how frequent the state variables are sampled). This allows recovery from, say, an infrastructure resource failure in a timely manner.

2.4 Management module for reconfigurable services (RSMM)

In a basic form, the RSMM maps client-initiated service requests onto a specific protocol at network infrastructure level. For this purpose, the RSMM maintains the binding information for various services provided by the network infrastructure. Our focus is on the additional functionalities desired of the RSMM: service monitoring and coordination of client adaptation, to support dynamic settings

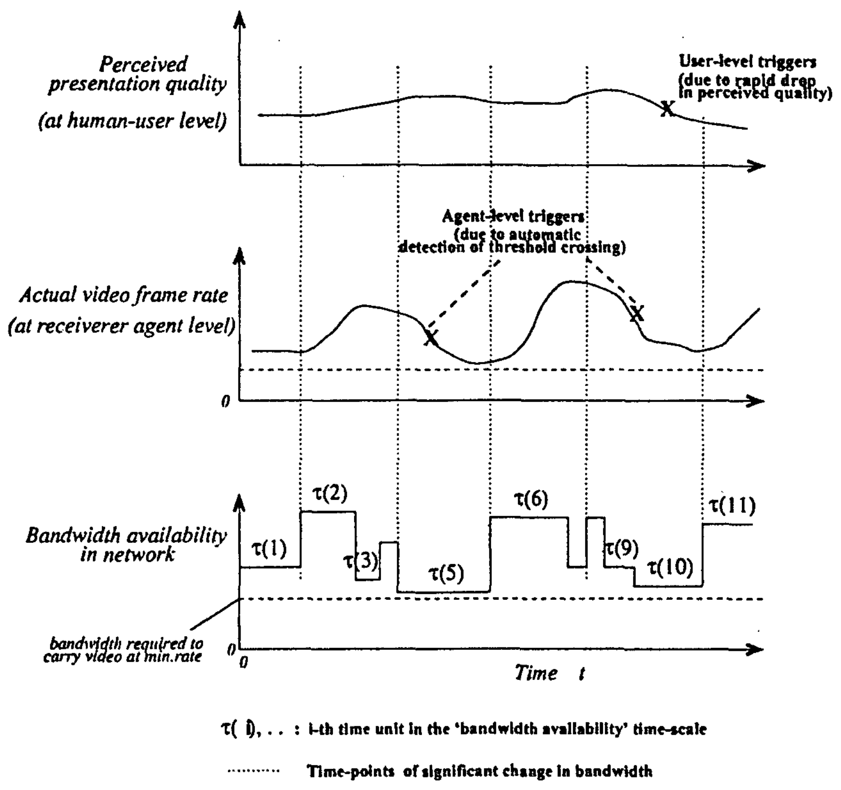

Perceived presentation quality (at human-user level)
User-level triggers (due to rapid drop in perceived quality)
Agent-level triggers (due to automatic detection of threshold crossing)
Actual video frame rate (at receiverer agent level)
0
Bandwidth availability in network
τ(1)
τ(2)
τ(3)
τ(5)
τ(6)
τ(9)
τ(10)
τ(11)
bandwidth required to carry video at min.rate
0
0
Time t
τ(i), . . : i-th time unit in the 'bandwidth availability' time-scale
Time-points of significant change in bandwidth

Fig. 2. Time-scales of parameters at various layers of video distribution network

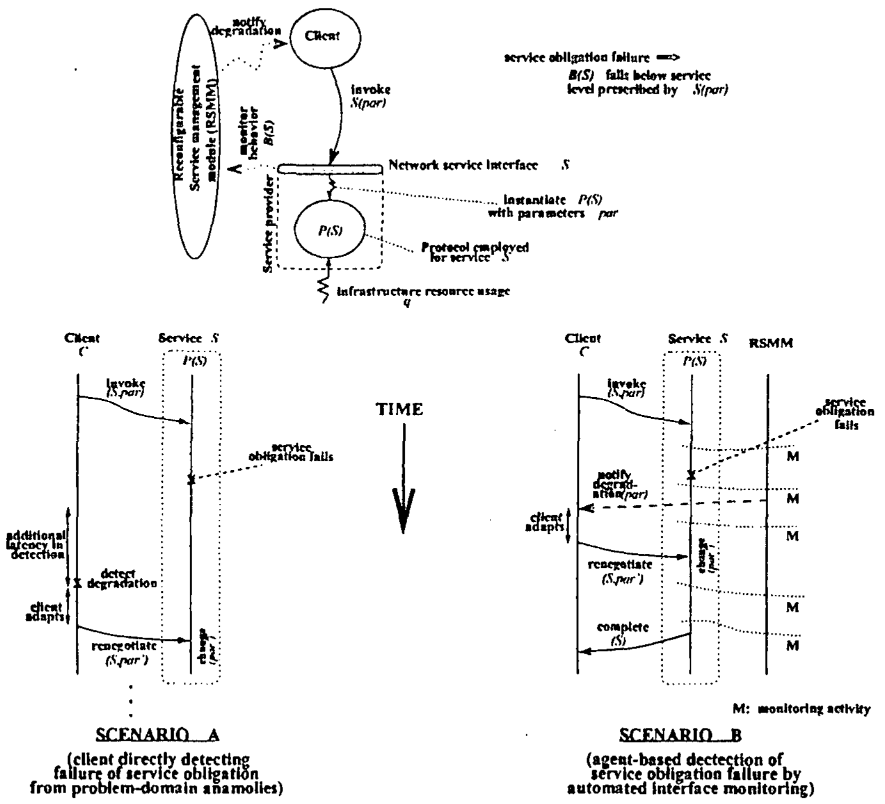
where changes and/or outages in infrastructure resources may occur at various points in time[4]. The RSMM support for service monitoring enables client adaptations to occur at computational time-scales. These cases are illustrated by <u>scenario-A</u> and <u>scenario-B</u> respectively in Figure 3. Without RSMM support, detection of service degradation by clients (and a subsequent adaptation) is possible only over problem-specific time-scales.

Suppose the client actions to deal with service degradations are expressible in a closed-form involving application-supplied functions. Here, the RSMM agents at the client and service sites can coordinate with one another to reconfigure the application over computational time scales (e.g., reducing the video send rate

[4] In comparison with isolating the network subsystems based on observed outages in aggregated network elements [9], our work focuses on application-level adaptations to deal with such outages.

in a 'multiplicative decrease' form during network congestion). Otherwise, the RSMM may signal an exception that triggers client-level recovery over problem-specific time scales (e.g., human intervention to reduce the video window size and/or switch to a low quality encoding [6]).

To monitor service-level compliance to client expectations, the RSMM instantiates service-level meta-data dissemination protocols with parameters pertaining to the problem-domain. The monitoring and control roles of RSMM should themselves be independent of the service-specifics (at meta-level), so that they can be employed across diverse problem-domains — through software re-use[5].



Reconfigurable Service management module (RSMM)
notify degradation
Client
invoke S(par)
monitor behavior B(S)
service obligation failure ⟹ B(S) falls below service level prescribed by S(par)
Network service interface S
Service provider
instantiate P(S) with parameters par
P(S)
Protocol employed for service S
infrastructure resource usage q
Client C
Service S
P(S)
invoke (S,par)
service obligation fails
additional latency in detection
detect degradation
client adapts
renegotiate (S,par')
change (par')
TIME
RSMM
M
notify degradation(par)
complete (S)
M: monitoring activity
SCENARIO A
(client directly detecting failure of service obligation from problem-domain anamolies)
SCENARIO B
(agent-based dectection of service obligation failure by automated interface monitoring)

Fig. 3. A high level view of reconfigurable network service offering

2.5 Management view of sample network application: CDN

A 'content distribution network' (CDN) server hosts the content pages of an information base — e.g., an electronic shopping catalogue. With a large number

[5] A service interface description should capture the 'flow of time' — so that 'time scale' can be expressed as a parameter [4].

of clients trying to access various parts of the information (or, pages), the server maintains copies of pages at multiple nodes of the distribution network (such as in AKAMAI) [10]. When a client (say, a web browser) accesses the CDN server for a page, the request is forwarded to the nearest node containing this page for downloading. Keeping copies of the pages at multiple nodes in the network (i.e., proxy replicas) increases access performance and offers higher availability of the information base. The attributes prescribed by CDN clients may include the tolerances to page access latency and missed deadlines on page delivery.

There are two elements of the protocol mechanisms: i) placement of replicas in the network, and ii) update policies for keeping the replicas consistent[6]. See

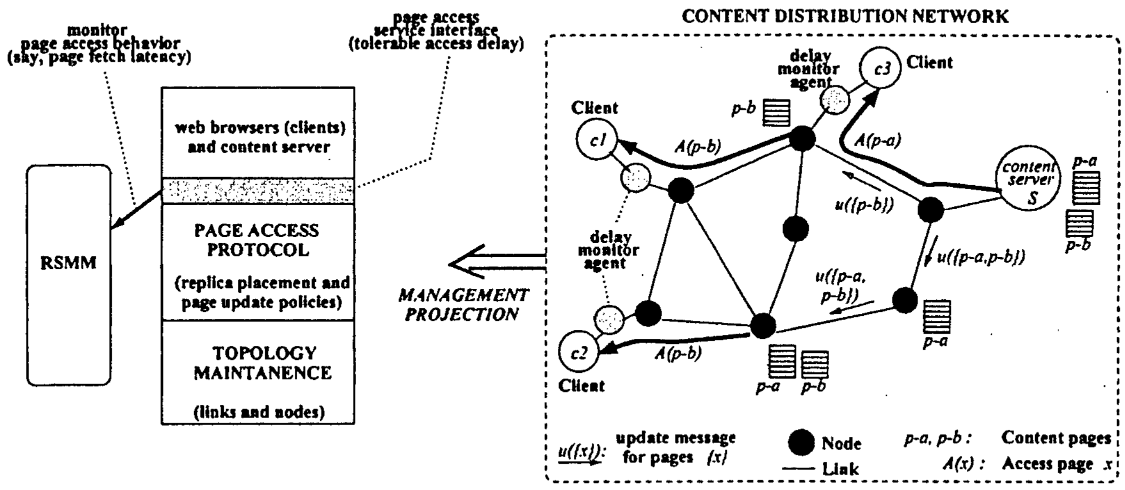


Fig. 4. Management view of a content distribution network

Figure 4 for illustration. The geographic spread-out of replicas relative to the location of clients in the network and the page update policies to keep the copies consistent determine the latency in fetching pages by a client. The policies for replica placement and page updates are chosen by the CDN protocol[7].

The protocol mechanisms (i) and (ii) constitute the 'resources' in terms of our service management model. The agents placed at client nodes continuously

[6] Page update policies include correcting the copy at a node when a client attempts to access this copy (client-driven updates) and correcting the copies at various nodes whenever the server changes its master copy (server-driven updates).

[7] In the management view, a CDN client need not be an end-user of content pages. The client may possibly represent an access box feeding pages to a community of customers in a geographic area. In the latter case, the access box may derive its 'access latency' prescription to the SP from an aggregation analysis of the individual customer needs. The QOS composition mechanisms employed by the access box (say, for customer pricing) are however outside the purview of a management model.

monitor the access latency, so that the client nodes can adapt their access behavior — such as removing a page from their access set if it incurs excessive latency and/or stipulating tighter controls on (i) and (ii).

As can be seen, a management view delineates the protocol mechanisms employed in the problem-domain and the generic monitoring and control tools that interwork with these mechanisms. In terms of this view, we now describe our 'function'-based service model and the underlying management techniques.

3 Our 'function'-based model of services

We employ an *application-oriented functional* approach to managing network services. It underscores the theme: reconfigurability of network services.

3.1 Service normalization

Figure 5 shows an architectural view of how different types of services can be

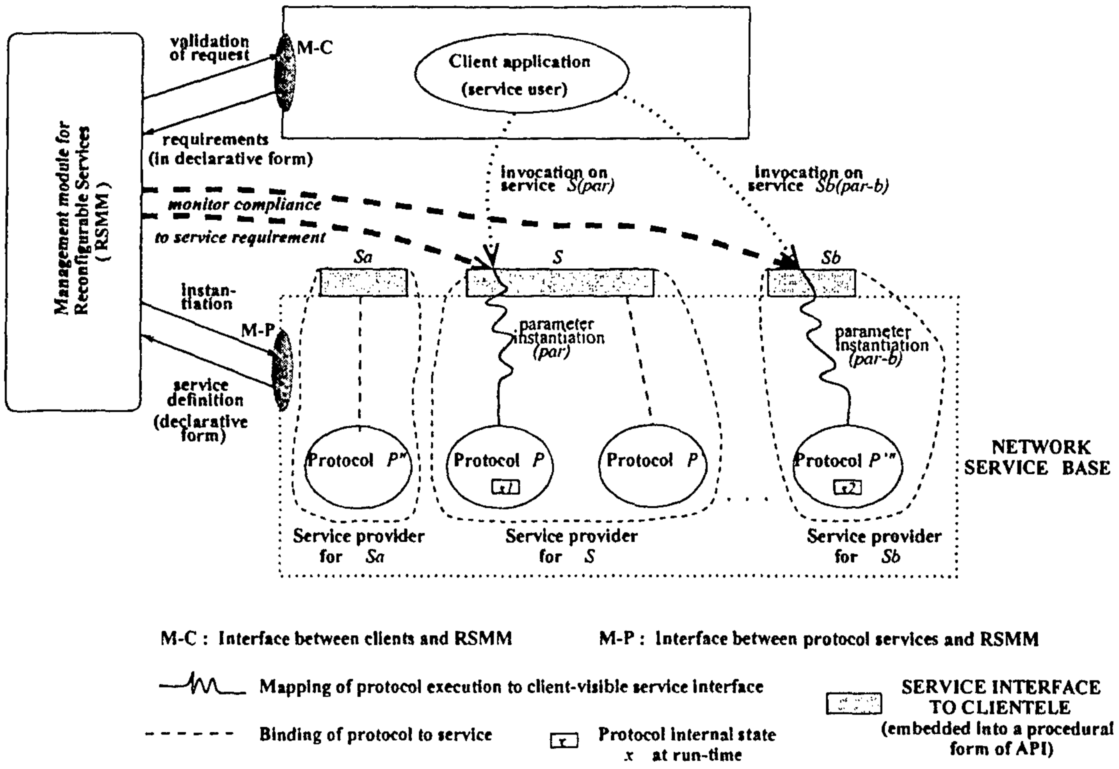

Management module for Reconfigurable Services (RSMM)
validation of request
M-C
Client application (service user)
requirements (in declarative form)
monitor compliance to service requirement
invocation on service S(par)
invocation on service Sb(par-b)
Sa
S
Sb
instan-tiation
M-P
service definition (declarative form)
parameter instantiation (par)
parameter instantiation (par-b)
Protocol P''
Protocol P
x1
Protocol P
Protocol P'''
x2
Service provider for Sa
Service provider for S
Service provider for Sb
NETWORK SERVICE BASE
M-C : Interface between clients and RSMM
M-P : Interface between protocol services and RSMM
Mapping of protocol execution to client-visible service interface
Binding of protocol to service
Protocol internal state x at run-time
SERVICE INTERFACE TO CLIENTELE (embedded into a procedural form of API)

Fig. 5. Architectural view of our service management system

supported through a general-purpose service management system.

First, when different protocol modules $P(S), P'(S), \cdots$ are capable of providing a given network service S, the externally visible behaviors of $P(S), P'(S), \cdots$ are represented through a set of features common to S. This ensures that the choice of a protocol module by the SP to handle a given client request on S and the use of service-specific functions to manage S are transparent to the RSMM. At the client-service interface level, the differences in internal mechanisms employed by $P(S), P'(S), \cdots$ are not visible to the clients. Second, with a multitude of network service offerings $S_1, S_2, \cdots$ to be managed by the RSMM, it is desirable that the behavior descriptions of $S_1, S_2, \cdots$ be captured through a uniform meta-language. This allows the RSMM to employ the same set of management tools across various types of network services, by simply instantiating the tools with service-specific parameters.

The above goals require that service behaviors are representable in a canonical form, in terms of the notations allowed by a generic service description language. Towards this end, the protocols $P(S), P'(S), \cdots$ expose a common *interface state* $\Psi(S)$ to clients through the SP, denoted as:

$$\Psi(S) = \{\alpha_i\}_{i=1,\cdots,k \text{ for } k \geq 1},$$

where $\alpha_1, \cdots, \alpha_k$ are state variables. From a client's perspective, the α_i's are the service attributes exported by S that can be compared against 'values' supplied by the client to determine if S meets the behavioral expectations.

3.2 Protocol-to-service mapping

A protocol $P(S)$ realizing the service S maintains a state s, which is a reflection of the infrastructure resource usage by the internal mechanisms of $P(S)$. At the service level, the interface state $\Psi(S)$ is a mapping of the form:

$$\Psi(S) = \gamma_{(P,S)}(s)$$

prescribed by the SP. Consider the example of a 'virtual circuit' service implemented by an ack-based window protocol over a network, referred to as WIN(VIRT). A range of achievable data transfer rates over the 'virtual circuit' can be prescribed by a client (i.e., data sender/receiver). Infrastructure resources are the send/receive window w, and the link capacity c and error probability e. A client-visible interface state is the link utilization, given as:

$$\Psi(\text{VIRT}) = \gamma_{(\text{WIN,VIRT})}(\{w, c, e\}),$$

where $\gamma_{(\text{WIN,VIRT})}$ may be an analytical formula for computing the link utilization α_1 in terms of the infrastructure parameters w, c and e (among others)[8] [11]. The $\gamma_{(\text{WIN,VIRT})}$ is encapsulated into an 'applicative' function supplied by

[8] The achievable link utilization is given by: $\alpha_1 = \frac{1}{1+\frac{1}{w}+\frac{(\bar{l}-1).T.R}{w.C}}$, where $\bar{l}$ is the mean number of transmissions per packet (expressed in terms of e, the timeout period T for retransmissions, and the packet size R).

the 'circuit' provider. In general, $\gamma_{(P,S)}$ depicts a static mapping relation between the protocol internal state s and the client-visible interface state $\Psi(S)$.

In legacy systems however, an explicit representation of the protocol internal state s may not be feasible (e.g., 'available bandwidth' on a TCP connection). In such cases, the SP may declare one or more of the service attributes as ***unspecified***, i.e., these attributes can neither be assigned 'values' by clients nor be exported as 'indicators' of network conditions by the SP. Here, the $\gamma_{(P,S)}$ is simply a stub library to implement the procedural hooks to the service module.

Given our state-oriented interface characterizations, a language like JAVA is more suitable for service-independent interface descriptions (in comparison to 'data structure'-oriented languages like XML [12]).

3.3 Service-level behavioral descriptions

The requirement of a generic behavioral interface can be met with a 'function'-based model of service provisioning that involves the RSMM and the client and service modules. We use modular decomposition principles to structure these functions and the flow of meta-information between them. Refer to Figure 5.

The externally visible behavior of S can be described as a set of generic logical relations on the interface state components $\alpha_1, \cdots, \alpha_k$:

$$L(\Psi(S)) \equiv \Phi(\{o_1(\alpha_1), \cdots, o_k(\alpha_k)\})$$

where $o_1, \cdots, o_k$ are 'applicative functions' that produce boolean results based on the values of $\alpha_1, \cdots, \alpha_k$ respectively and $\Phi(\cdots)$ depicts logical relations: AND, OR, and NOR. Such a behavior description allows the RSMM tools to be independent of the problem-domain S pertains to.

Consider the example of a video distribution service (VDIST). The sustainable frame rate and the inter-frame delay are the interface state components. $L(\Psi(\text{VDIST}))$ may capture a condition: 'frame rate is no less than *vrate*' and 'frame-delay is no more than *vdel*'. This condition may be represented as:

$$L(\Psi(\text{VDIST}) \equiv [> (\text{FRATE}, \mathit{vrate}) \wedge \; < (\text{FDEL}, \mathit{vdel})],$$

with '>' and '<' being 'applicative' functions to compare with the values *vrate* and *vdel* respectively, as prescribed by the client (for use by the RSMM); FRATE and FDEL are the names of service features that allow client-level handle on the frame rate and frame delay respectively.

As can be seen, the service interface state may be derived through static mappings of protocol internal states, whereupon service behaviors may be determined by exercising logical relations among the interface state components. The extraction of service behaviors from the interface state forms a core part of the monitoring functionality of RSMM.

Largely, our RSMM acts as a *broker*, liaisoning clients with services, based on client-specified requirements and published service capabilities. The actual provisioning of a requested service is itself done directly by the SP. This is different from the 'contract-bidding' based management model suggested in [17] which requires service-specific procedures to be rigidly integrated into the model.

4 Related paradigms for managing distributed networks

We project our service management function in the light of currently prevalent management approaches: i) resource-level monitoring, and ii) 'SNMP'-based signaling. Our comparison is on the programming model of service management.

4.1 Resource-level monitoring

The ReMoS system [13] provides a query-based interface to obtain information about resource availability in a form meaningful for the SP to support application-level information flows. Likewise, the QOS Broker System (QBS) [14] provides for QOS specification, resource reservation, and resource monitoring. The QBS allows prescribing QOS violations that can be detected by 'sensor' objects (e.g., video frame delay jitter threshold of .15 *msec*). The scope of our work, when cast in the above lights, is in the SP-level mapping of flow specs onto network resource parameters.

The Rutgers Environment Aware API [15] provides for application adaptation through asynchronous event delivery, where an event is prescribed at an appropriate abstraction level along with a handler for that event type. Though the Rutgers API caters to a variety of network applications, its usefulness has been studied mainly to provide adaptation in relation to network-centered parameters (such as link bandwidth and security). In contrast, our model unifies the management of both information networks and data networks through a canonical and meta-level service-oriented interface. We extend the notion of infrastructure resources to cover beyond the traditional network parameters.

The RSMM part of SP may employ, say, the ReMoS primitives or the Rutgers API to monitor the symptoms of resource failures over selected time-scales. For instance, the ReMoS procedures for statistical estimations can be employed by the RSMM for smoothing and/or filtering of monitored parameters. Likewise, the QOS specs from applications can be signaled to the SP using the ReMoS primitives. The RSMM may also adopt the QBS methods to prescribe when an increase or a decrease in resource usage should take place. In contrast with such QOS specifications and mapping, the control actions of SP for QOS management are problem-specific (e.g., determining the proxy placement in CDNs). So, the SP consists more of 'resource adaptation' functionalities mediated through the RSMM. Our service-level management model reflects this emphasis.

4.2 SNMP-based network monitoring approach

SNMP defines a set of APIs that can be invoked at a network management station, and also the underlying (signaling) message exchanges between the management station and the agents executing at target protocol sites [16]. A 'sniffer' tool is provided that examines all packets exchanged through the protocol being monitored, without disrupting their flow. One useful SNMP feature is a prescription of packet filters based on 'logical expressions' that select which packets are captured for further analysis (e.g., all packets from an IP host to a specific 'ethernet interface' address). An event table prescribes when a notification or an alarm, is to be sent to the management station (e.g., when the number of packets destined to a given host exceeding a limit).

Packet selection criteria and the functions necessary to prescribe events therefrom are oriented towards traffic and fault analysis at various network elements. For example, the number of ICMP packet losses suffered at a router node exceeding a limit may be treated as indicative of a crash of this node. Arbitrary packet selection criteria can be loaded into 'filter tables' maintained by the network management system, along with a provision of basic functions to operate on the filtered streams of packets (e.g., computing the probability of packet loss). In contrast, our approach is towards behavior analysis of network subsystems as seen by applications through service interfaces.

In SNMP, the 'network management' user basically selects a function to operate on a packet stream from a menu of functions (possibly, through a GUI) hardwired into the management system. From the perspective of service programming, the presence of human user in the monitoring-reconfiguration loop makes pushes the SNMP model to a lower level, in comparison to our approach.

We now provide, as a case study, a management-oriented description of how service-level monitoring and control can be realized in CDNs — c.f. section 2.5.

5 Case study of reconfiguration management in CDNs

We first capture the service-level behavior, i.e., changes in page access latency with respect to the degree of page replication — as shown in Figure 6. We then describe the meta-activities of RSMM to adjust the page replication at run-time.

The page access latency PLAT, which is a client-visible attribute, is influenced by the replication mechanism that allows the nearest and most-recent copy of a page in the network to be accessed. The protocol-level mechanism depicts the exercising of infrastructure resources, namely, 'sync' message exchanges and traversal of data over network links.

The internal state of a protocol that realizes the CDN service is the current placement of replicas $repset_{cur}$ and the link delays between nodes $ldel_{cur}$. The SP may provide a mapping function:

$$plat(i) = \gamma_{(U,\mathrm{CDN})}(repset_{cur}, ldel_{cur}),$$

to yield the value of PLAT in an i^{th} sampling interval. The function may be specific to an update policy U on pages employed by the CDN protocol (i.e., client-driven or server-driven or page lifetime-based).

We consider four types of operations provided by the CDN SP for use by the RSMM to control the replica placement:

$$\begin{aligned}
&plat(obs) = \mathrm{smooth\_latency\_samples}(\{plat(i)\}_{i=1,2,\cdots});\\
&plat_{new} = \mathrm{set\_latency}(plat'_c, plat(obs));\\
&repset_{new} = \mathrm{get\_replica\_placement}(plat(obs), plat_{new});\\
&\mathrm{change\_replica\_placement}(repset_{new});
\end{aligned}$$

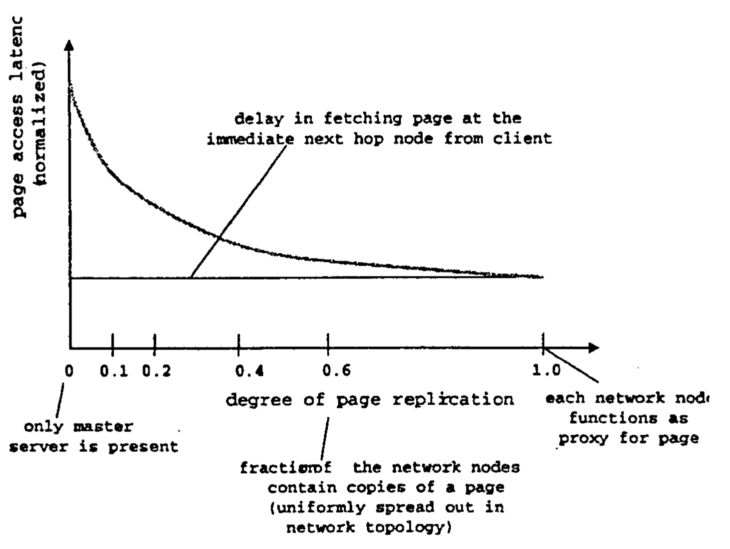


Fig. 6. Quantitative behavior-oriented study of CDN service

where $\{plat(1), plat(2), \cdots\}$ are the time-chronological sequence of samples of PLAT from which a smoothed observation $plat(obs)$ can be extracted and $repset_{new}$ depicts a new placement of replicas on the CDN nodes to bound PLAT to less than $plat_{new}$. The 'smooth_latency_samples' operation may simply be an averaging mechanism over a time-scale meaningful to the 'content distribution' application. The condition $> (plat(obs), plat_c)$, i.e., the monitored 'value' $plat_{obs}$ exceeding $plat_c$, may cause the RSMM to trigger a change in replica placements, where $plat_c$ is the currently prescribed value for PLAT.

The 'set_latency' operation may compute $plat_{new}$ as, say, $[plat'_c - \Delta \times (plat(obs) - plat'_c))]$, where $plat'_c$ is a modified limit on latency prescribed by the client and Δ is a parameter used by the SP for 'control-theoretic' stability of the changes in latency. The 'get_replica_placement' function may employ the $\gamma_{(U,\mathrm{CDN})}$ function in determining $repset_{new}$ that will cause the latency to not exceed $plat_{new}$. The 'change_replica_placement' function is specific to the update policy employed by the CDN protocol. Suppose a server-driven update policy is employed. If $repset_{new} \supset repset_{cur}$, the replicas $(repset_{new} - repset_{cur})$ are initialized to the current page content, and then installed at the designated CDN nodes. If $repset_{new} \subset repset_{cur}$, the replicas $(repset_{cur} - repset_{new})$ are removed from the concerned CDN nodes. Other cases can be a combination of these mechanisms.

The operations listed for a CDN service are supplied by the SP as 'applicative' functions that can be invoked by the RSMM. The client prescribes the monitoring conditions through 'applicative' functions: the '>' relation on latency values, for use by the RSMM[9].

6 Conclusions

When developing distributed network services, challenges arise due to sub-system level failures and/or changes that may dynamically occur and can, in turn, affect the application functionality. How such sub-system level events impact applications is specific to the problem-domain. In this paper, we described an automated and generic management tool that enables the client application to reconfigure whenever there is a service change or failure.

To support the integration of service management tools into distributed network development environments, we employed a paradigm founded on modular decomposition principles. In this paradigm, a service may be provided by a protocol module through a generic interface, with the client application instantiating this module with a desired set of parameters. The service-level management module (RSMM) maintains the binding information for various network services that can be provided through the infrastructure resource usage. The RSMM supports a highly dynamic setting, such as changes and/or outages in service provisioning occurring at various points in time and in different time-scales. The RSMM effectively 'brokers' between clients and services to coordinate their interactions in a flexible and configurable manner.

Our paper described a new model of service provisioning: a 'function'-based decomposition of service components. The model separates a service interface to clientele from the details of protocol modules that actually provide the service. Quality requirements may be prescribed at a service interface, in the form of logical relations on the service attributes. Client applications may instantiate a network service, along with a prescription of the required service quality. The model allows dynamic reconfiguration from one set of parameters to another, in case a service degradation occurs at run-time. For this purpose, the RSMM incorporates a monitoring functionality that checks for compliance to the prescribed quality requirements at run-time. The paper also described the case study of a network application: CDN.

The 'function'-based structuring and the service-neutrality that underscore our service model offer the flexibility of service growths and the extensibility of supporting diverse client requirements. This is much broader in scope, when compared to the existing management models based on SNMP and OSI-TMN. Our model can be implemented in an appropriate object-oriented programming language that offers the capability of dynamically dispatchable protocol code (such as JAVA) [19]. It can thus facilitate the rapid development and deployment of network services (say, using the CORBA Middleware API [4]).

[9] [18] provides architectural frameworks to encapsulate 'policy' functions in systems for managing service-level infrastructures.

References

1. S. Erfani, V. B. Lawrence, and M. Malek. **The Management Paradigm Shift: Challenges from Element Management to Service Management.** In *Applications, Platforms, and Services*, Bell Labs Journal, 5(3), pp.3-20, July-Sept. 2000.
2. M. Subramanian. **Telecommunications Management Network.** Chap. 11, *Network Management: Principles and Practice*, Addison-Wesley Publ. Co., 2000.
3. M. Horstmann and M. Kirtland. **DCOM Architecture.** MSDN Library, *http://www.msdn.microsoft.com/library*, July 1997.
4. V. Fay-Wolfe and et al. **Real-time CORBA.** In *IEEE Transactions on Parallel and Distributed Systems*, 11(10), pp.1073-1089, Oct. 2000.
5. A. S. Tanenbaum. In **Switching: The Telephone System.** Chap. 2., *Computer Networks*, Prentice-Hall Publ. Co., pp.130-134, 1996.
6. C. Diot, C. Huitema, T. Turletti. **Multimedia Applications Should be Adaptive.** In Proc. *HPCS'95*, Mystic (CN), Aug. 1995.
7. K. Ravindran and R. Steinmetz. **Object-oriented Communication Structures for Multimedia Data Transport.** In *IEEE Journal on Selected Areas in Communications*, 14(7), pp.1360-1375, Sept. 1996.
8. K. Ravindran and K. K. Ramakrishnan. **Feature-based Service Specification in Distributed Systems.** In proc. *ICDCS'91*, Arlington (TX), May 1991.
9. V. Sethaput, A. Onart, and F. Travostino. **Regatta: A Framework for Automated Supervision of Network Clouds.** In Proc. *OPENARCH 2001*, Anchorage (AK), pp. 104-114, April 2001.
10. J. Chase, S. Gadde and M. Rabinovich. **Web Caching and Content Distribution: a View from the Interior.** In 5th Intl. Workshop on *Web Caching and Content Delivery*, Lisboa (Portugal), 2000.
11. M. Schwartz. **Telecommunication Networks: Protocols, Modeling, and Analysis.** Addison-Wesley Publ. Co., Nov. 1988.
12. M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster. **UIML: An Appliance-Independent XML User Interface Language.** In proc. *Eighth International World Wide Web Conference*, Toronto (Canada), 1999.
13. T. Dewitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, J. Subhlok, and D. Sutherland. **ReMoS: A Resource Monitoring System for Network-Aware Applications.** In *Tech. Report*, CMU-CS-97-194, Dec. 1997.
14. M. Katchbaw, H. Lutfiyya, and M. Bauer. **Driving Resource Management with Application-level Quality of Service Specifications.** In Proc. *1st Intl. Conf. on Information and Computation Economies*, ICE98, Oct. 1998.
15. B. Badrinath, A. Fox, L. Kleinrock, G. Popek, P. Reiher, and M. Satyanaranyanan. **A Conceptual Framework for Network and Client Adaptation.** In *IEEE Mobile Networks and Applications*, 2000.
16. M. Subramanian. **SNMP Management RMON.** Chap. 8, *Network Management: Principles and Practice*, Addison-Wesley Publ. Co., 2000.
17. G. Kar and A. Keller. **An Architecture for Managing Application Services over Global Networks.** In proc. *INFOCOM'01*, IEEE-CS, Anchorage (AK), pp.1020-1027, April 2001.
18. D. Verma, M. Beigi and R. Jennings **Policy-based SLA Management in Enterprise Networks.** In *Res. Report*, IBM T. J. Watson Res. Center, 2001.
19. D. J. Wetherall, J. V. Guttag and D. L. Tennenhouse. **ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols.** In Proc. *IEEE OPENARCH'98*, San Fransisco (CA), April 1998.

Appendix 2

Programming Models for Behavioral Monitoring of Distributed Networks

K. Ravindran *
Department of Computer Science
City University of New York, USA.
E-mail address: *ravi@cs.ccny.cuny.edu*

Abstract

We provide a specification methodology to monitor the system-level compliance to properties deemed as critical for functioning of a distributed information system (DIS). Since information attacks due to external intrusion and/or component failures manifest as deviations from critical behaviors expected of a DIS, violation of a specified critical property can be viewed as symptomatic of information attacks. Users can prescribe critical properties in the form of *event predicates*, which are boolean conditions on the externally visible interface state distributed among computation nodes and can be detected by distributed algorithms. Any target application can then plugged-in to our generic monitor tool to test for its compliance to critical functionality. With distributed object-oriented programming support (e.g., JAVA), our monitor can reduce the software development costs of a DIS: due to the ease of specification and maintenance.

* Part of the work on user-interface support for distributed monitoring is currently supported by AFRL-HE grant: F33615-03-1-6385 (2003). Part of the work on programming and algorithmic aspects was earlier supported by PSC-CUNY Award (2000).

1 Introduction

The goal of our paper is to provide a specification methodology that allows the monitoring of system-level properties deemed as critical for functioning of a Distributed Information System (DIS). Since an infrastructure-level attack and/or QOS degradation often manifests as a deviation from critical behaviors expected of a DIS, a violation of compliance to a specified critical property can be viewed as symptomatic of such failures. Based on this principle, users can prescribe critical properties in the form of *event predicates*, which are boolean conditions on the externally visible interface state distributed among computation nodes and can be detected by distributed algorithms. Since the information-level attributes of a DIS may dynamically change over problem-specific time-scales, the event predicates include a prescription of the 'flow of real-time' as meaningful in the problem-domain.

From a programming perspective, our specification methodology consists of two facets:

1. A meta-language which allows the interface behavior of a DIS to be prescribed through possible occurrences of events in the external environment of the application;

2. Incorporating the specification language in a reactive programming system that exposes the computation-level states symbolizing the occurrence of events.

These facets can be built into a generic monitor API that can be instantiated with problem-specific parameters and functions. Any target application can then be plugged-in to the monitor tool to test for compliance to its critical functionality requirements. A system developer can easily augment the monitor API with application-specific libraries to form a complete distributed programming system.

The realization of a generic monitor in distributed object-oriented programming languages (such as JAVA) can reduce the software development costs of implementing a DIS: due to the ease of specification, development and maintenance of the DIS.

The paper first begins with the generalized structure of a DIS to expose the underlying system-level issues. It then narrows down to our proposed specification methods for system monitoring.

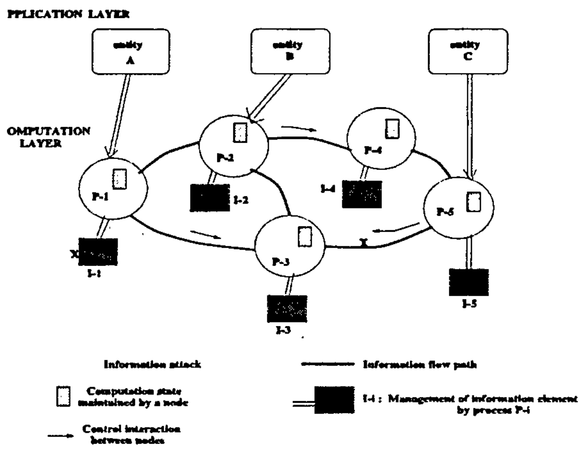


Figure 1: Bird's view of a DIS

2 Behavioral view of DIS

An information system for large scale real-time applications may be viewed as carrying out a distributed computation that interacts with external environments consisting of:

- Data sources that function as *sensors* to track the changes occurring in the application environment (e.g., ground vibration analyzers to detect seismic variations);
- Data sinks that function as *actuators* to map computation results onto specific effects in the application enviroment (e.g., a motor to position a radar unit).

Human users and/or intelligent autonomous agents at distinct geographic locations may be part of the DIS, interacting with one or more computation nodes to provide strategic decision support [1]. See Figure 1.

2.1 Why monitor system behaviors ??

The behavior of a DIS as a whole may be continually changing, sometimes to even below acceptable threshold levels of operations. This may be due to sub-system level components (viz., computation nodes and/or their interconnections) being forced to operate in a degraded mode because of intrusion by attackers, component failures and/or undesirable changes occurring in the external environment. Visible changes in a system behavior mean that the behavior is measurable in terms of concrete parameters of the system.

Consider, as an example, sensor devices deployed in battlefield terrain, say, to detect enemy movements. These sensors are prone to failures (such as loss of battery power to sensors). At a macro level, the reliability of inferences made about enemy movements, which depends on the accuracy and timeliness of sensor data, is a measurable property of the detection system. Some form of 'majority voting' technique (such as 'at least 10 sensors in the terrain have detected enemy movements) may be employed to increase the quality of inferences. The decision-making by commanders (say, to initiate attacks on enemy positions) may itself may be based on such a quantitative assessment of the 'inference reliability' (or confidence level) parameter[1].

With such failures, it may often be the case that a failure can be detected after some damage has been inflicted upon the information. Since a contaminated information affects the visible behavior of a DIS in some way, we need to instrument mechanisms for detecting deviations from the expected behaviors. The issue then is on sufficiently early detection to prevent any further damage.

2.2 Critical system properties

The acceptability of a system may be based on whether its observed behavior meets certain criteria that are absolutely essential in order for it to function without interruption. A prescription of these essential criteria may be deemed as a *critical property* of the system. A critical property may be associated with specific *symptoms* that can be observed by a management module in the computation subsystem.

Suppose, in the earlier example of sensors in a 'digital battlefield', it is stipulated that the confidence level in the reliability of 'enemy movement' information exceed a certain threshold, say 70%, in order to initiate a pre-emptive strike on enemy positions. When the number of active sensors falls below the corresponding 'majority' figure in the number of sensors detecting enemy movement (say, 70% of the sensors), the detection system can no longer be able to supply information at the critical level of reliability.

In a broad sense, critical properties may cover many aspects of system behaviors pertaining to quality of information delivery (QOS) to the application.

[1]Changes occurring at system hardware and/or software level may be abstracted into changes occurring at the information level. In the example of sensors, the effect of battery power-down may be subsumed into an observable reduction in the confidence level on 'enemy movement' information derived from sensor data. So a DIS should be modularized in such a way that macro-level anamolies of interest can be inferred from the easily observable behavior of system components.

Here, quality is measurable through macroscopic indices capturing the timeliness, accuracy and/or precision of the information presented. Some instances where information quality can suffer due to incursions from the external environment are:

- Transport quality offering in data network settings (e.g., an increase in network delay jitter causing frequent glitches in a remote video presentation);
- Faults occurring in system components (e.g., the crash of a replica node that reduces the 'data availability' in an information repository);
- Malfunctions in subsystem-level components and/or their interconnections (e.g., failure of communication links in a distribution network that reduces the connectivity among nodes).

The above categorization is not exhaustive though. It is up to the application to determine what constitutes a mandatory property that must be met during a computation.

Regardless of the criteria in enunciating a critical system property, it is necessary to continually monitor a system behavior, so that reasoning about when the system meets its critical expectations and when it fails to meet can be made. Due to the real-time nature of information, detection of critical property violations that may arise from information attacks is also time-sensitive.

3 A real-time programming view

Critical property violations are often symptomatic of information failures. The programming-oriented aspects to be considered in monitoring the behavioral symptoms are described in this section.

3.1 'event-action' based framework

The behavioral symptoms of a DIS should be programming-level projections of the anamolies occurring in the problem-domain (i.e., deviations from expected critical behavior) that can be observed across the management interface between decision-making entities in an application and the underlying computation sub-systems. The management interface should be programmed to notify the application entities when the symptomatic parameters fall outside the acceptable range of values. In the example of sensor failures, the 'inference reliability' parameter may be continuously tracked on a management console through a GUI, with the GUI raising, say, an audio-visual alarm when the parameter falls below 70%. Such a'call-back' notification may trigger a recovery action by the application entities (e.g., the commander at the console replacing the faulty sensor banks with good ones, or resorting to alternate means of inferring enemy movements).

3.2 Faster time-scales of monitoring

The parameters monitored by the management module should vary at much smaller time-scales than the anamolies in the problem-domain they symptomize. This is because the problem-domain has a much slower dynamics in comparison to the dynamics of the program-level parameters chosen to map to the problem-domain. In the earlier example of sensor failures, changes in the 'inference reliability' parameter can usually be detected in the time-scale of running a distributed algorithm on computation nodes that conducts the voting among sensors, which is much smaller than the time-scale of the effects sensor failures have on the 'battlefield' operations (such as unnoticed enemy movments). The[2] information dissemination algorithms implemented by application entities typically interpret the monitored program-level parameters at their native time-scales, in order to recover from, say, a mal-function in the problem-domain in a timely manner.

The above requirements of system-level monitoring should be reflected in the specification methods that describe system-level critical behaviors.

3.3 Modular structuring of monitor

We break up a DIS into a physical system that is being controlled, and a set of sensors and actuators through which the physical system 'connects' with a computation sub-system. The latter consists of an *information collector* that pre-processes the 'sensed' data from the physical system, a *distributed monitor* that examines the sensor data to ascertain the system compliance to prescribed critical properties, and a *repair controller* that generates recovery actions in the form of driving the actuators to perturb the physical system in a way to ameliorate any deviation from ex-

[2]Consider, as another example, a video distribution application. A degradation in the quality of presentation due to a reduction in video frame rate is perceived by receivers much slowly, in comparison to the time-scale over which the frame rates actually fluctuate (say, due congestion in the video transport network).

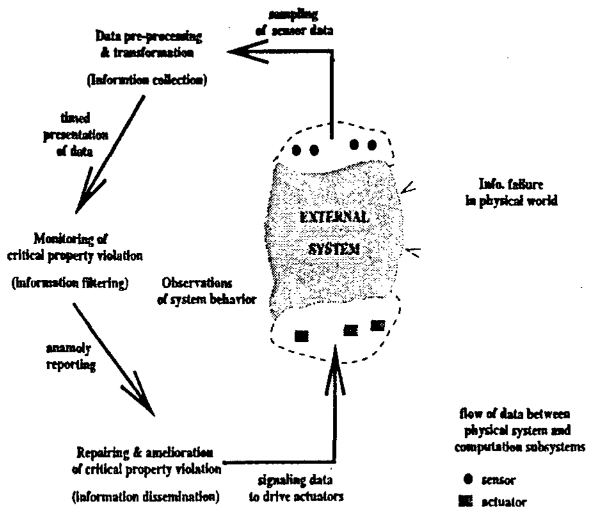

sampling of sensor data
Data pre-processing & transformation
(Information collection)
timed presentation of data
Monitoring of critical property violation
(information filtering)
Observations of system behavior
EXTERNAL SYSTEM
Info. failure in physical world
anamoly reporting
flow of data between physical system and computation subsystems
Repairing & amelioration of critical property violation
(information dissemination)
signaling data to drive actuators
● sensor
■ actuator

Figure 2: Monitor-based structure of DIS

pected critical behavior. The overall goal of information processing algorithms implemented in the computation subsystems is to keep the operating point of the DIS at a desired level. See Figure 2. As can be seen, the 'sensors' and 'actuators' may be abstract in nature delegated with a computational intelligence to carry out pre-processing and/or transformations on the input data and output results.

Since a bulk of the monitoring functionality is generic in nature, a general purpose monitor module that can be instantiated with problem-specfic parameters, functions and time-scales. Our proposed approach is based on this premise.

3.4 Scope of current programming models

The event monitoring is an instance of the 'distributed state construction' problem [2]. Currently available programming models for 'classical' real-time control (as in avionics and industrial applications) often employ an *integrated* approach, where the monitoring and control functionality are blended into the computation modules that constitute the target system being controlled [3]. The monolithic structure of computations underlying an 'integrated' programming view does not allow an easier separation of the application-independent and application-specific processing activities across the (management) interface between computations and external environment. Consequently, any behavioral description of a DIS needs to be hardwired into the computational procedures realizing the system. Such behavioral descriptions cannot be extracted out from the overall system specification for the purpose of explicit representation and manipulation by general purpose modules. This makes it difficult to re-use the functionality built into one system for another system, which manifests as redundancies in distributed software development.

Thus, we need a modular approach whereby the computational views of the physical system structure and the temporal behaviors of the system can be captured through generic programming modules instantiated with problem-specific parameters. Essentially, the monitor-based structure treats the 'sensors' and 'actuators' as encapsulated computational objects and the computation sub-system as an I/O automaton interfacing with these objects. Data from the 'actuators' cause state transitions in the computation that symbolize changes in the physical system. The 'sensors' track the state changes occurring in the physical system, with the monitor serving as a filter to detect only the meaningful changes that need to be notified to the 'repair' module. Such a computational view allows incorporating dynamic adaptivity in the functionality offered by a DIS.

4 Specifying critical properties

From a programming perspective, critical properties constitute invariants of a system. Occurrences of specific system-level events may constitute *symptoms* of situations where critical properties are violated. So system monitoring basically amounts to detecting the occurrences of specific events associated with these properties[3].

4.1 Timing attribute of information

An information element $I_i|_{i=1,2,\ldots}$ maintained by a DIS may be given by a pair: $(I_i, val_i(t))$, where $val_i(t)$ is the value assumed by the element I_i at time t. Because of changes occurring in the external environment, the information elements are in a perpetual mode of changing (over time). Certain changes in val_i's may indicate a significant deviation in the external environment relative to its current operating point, which may be deemed as an 'event'.

[3]'event detection' is a first-class abstraction, the complexity of which is encapsulated behind well-defined procedural interfaces [4]. Monitors are then objects that detect the occurrences of events in the problem-domain.

We may prescribe a time scale ϵ_i over which changes in I_i occur, i.e., ϵ_i may be viewed as an average time interval between changes in I_i. For example, a noticeable change in aircraft coordinates provided by a tracking radar may not occur any more frequent than once in 5 *sec*. We refer to this property as 'element I is ϵ-stable'. The ϵ parameter may represent the time interval for sampling the sensor data emanating from external environment.

Suppose an event pertaining to I_i is observed to occur at time t. An event handler starts at time $t' > t$ to carry out, say, a recovery action that may persist for a duration $\beta > 0$, where $(t' + \beta) < (t + \epsilon_i)$. In other words, the event handler should complete before, on the average, a next change in I_i occurs. The ϵ parameter depicts the time scale of event occurrences.

In the underlying execution of a distributed algorithm, the parameter $(\epsilon - \beta)$ may specify an upper bound on the time for collecting distributed snapshots of the system state.

4.2 Event predicates

In general, any type of critical property may be represented by a *condition* on the state variables $\Psi(S)$ exported by the external interface S that is visible to the application entities (i.e., users). An associated system event is said to occur when user-prescribed condition on the problem-specific state holds, i.e., a predicate $L(\Psi(S)) = \text{true}$ where $L(\cdots)$ is a logical formula. When the event occurs, it is understood that the system has indeed failed to meet the corresponding critical requirement. Some conditions may involve the state information at only one node, whereas some other conditions may involve many or all nodes.

An event e is said to occur at time t when a predicate *evnt_spec*(e) capturing certain type of changes in system interface state S becomes true in the interval $[t, t+\epsilon]$. These changes are observable by a distributed algorithm by sampling the interface state $\psi(S)$. Such event definitions may be given as:

$$evnt\_spec(e) \equiv L(\{(F_i, (\psi(S)_i = (A_i(t), k_i))\}_{i=1,\cdots,N}),$$

where F_i is an 'applicative' function that relates the observed value of an attribute A_i at time t with a constant k_i to yield a boolean result (such as $>$, $<$, $=$, $+$, $\cdots$). The functions $\{F_i\}$ may be hooked onto the computation as 'plug-ins' that can be invoked by the monitor to process the sampled sensor data as inputs[4].

The condition $L(\Psi(S))$ is supplied by a user to his/her local computation node. Once an event predicate is prescribed, a distributed 'event detection' algorithm (such as those described in [6, 7]) is executed by monitoring nodes $\mathcal{M}$ to evaluate the predicate at various time points using the instances of interface state $\Psi(S)$ maintained by the computation nodes $\mathcal{C}$ (where $\mathcal{C} \bigcap \mathcal{M} \neq \emptyset$). The event is deemed to have occurred when the predicate evaluates to true, whereupon, this result should be propagated to appropriate nodes $\mathcal{C}$ (so that a recovery may be initiated).

The occurrence of events will be determined by applying the logical formula $L(\Psi(S))$ on a composition of the states maintained by computation nodes $\mathcal{C}$[5].

4.3 Illustrative examples

Consider the sensing of temperature p in a boiler plant. Suppose a critical situation is stipulated as the rate of increase of temperature exceeding a threshold T_2 when the current temperature is above T_1 (this may trigger the opening of a coolant liquid valve so that an explosion can be avoided). This event may be expressed as:

$$evnt\_spec(\text{EXPLODE}) \equiv (avg(p) > T_1 \wedge \frac{dp}{dt} > T_2).$$

Suppose the valve which was opened needs to be closed when the temperature falls below a threshold T_1' $(< T_1)$ — which implies that $\frac{dp}{dt} < 0$ — to avoid a freezing[6]. This relation may be expressed as:

$$evnt\_spec(\text{FREEZE}) \equiv (\{(avg(p) < T_1'\} \wedge \{(\frac{dp}{dt} < 0)\})).$$

The physical device that samples p is embedded into the snapshot-taking mechanism which filters the timed samples from the device.

Consider the earlier example of determining the 'inference reliability' parameter for a bank of sensors that detect enemy movements on the battle terrain. Treating the bank of sensors as providing a 'surveillance service', a critical property of this service may be prescribed as:

$$evnt\_spec(\text{SENREL}) \equiv (N_s > 10 \wedge \frac{N_g}{N_s} > 0.7),$$

[4]Our functional approach to specifying interface behaviors achieves more modularity than the 'state transition' based approach employed in [5] for database-like heavy objects.

[5]Sensor data from external environment may be made available to $\{F_i\}$ through auxiliary measurement tools which, in a programming sense, are outside the monitor proper.

[6]That $T_1' < T_1$ implies a 'hysteresis' during decreasing trends of temperature, to avoid valve 'chattering'.

where N_s is the number of sensors in the bank and N_g is the number of non-faulty sensors. N_s and N_g constitute the programming-level state visible at the service interface, in terms of which the event depicting a drop in the inference reliability to unacceptable levels is prescribed. The underlying protocol mechanisms to determine N_g from a knowledge of the set of sensors in the bank (such as 'majority voting' among sensors) is insulated from this interface specification[7].

Once the above form of predicates is loaded into the event monitor algorithm, the latter mechanically takes a snapshot of the system state maintained at various nodes — $(p, \frac{dp}{dt})$ and (N_s, N_g) respectively. It then computes the applicative functions — '$avg(\cdots)$'/'$\frac{d(\cdots)}{dt}$' and '$>$'/'$\frac{(\cdots)}{(\cdots)}$' on p and N_s/N_g respectively — to evaluate *evnt_spec*(EXPLODE) and *evnt_spec*(SENREL), as the case may be.

4.4 Event causality in system behaviors

Every event monitored at the system interface level is associated with a logical time of occurrence, namely, processing the change in interface state caused by an event e in the context of that caused by an event e'. We denote this relation between e and e' as $e' \succ e$. Due to involvement of human elements in target computation, the events can be observed only in an order given by the $\succ$ relations among them (i.e., causal order) [8]. For instance, intrusion at a security classification level $j+1$ can occur only after that at a level j, where $j = 0, 1, 2, \cdots$ with increasing values of j indicating higher classification levels. This may be denoted as: intrusion(j) $\succ$ intrusion($j+1$). The causality relations between events are specific to the given problem domain (but may be expressed using a common notation).

Consider the earlier example of boiler plant temperature control. The *open_valve* and *close_valve* operations may be supported by a recovery module as part of the temperature control. Even though the enabling conditions for *open_valve* and *close_valve* are prescribed separately (by EXPLODE and FREEZE events respectively), there is an application-specific causal relationship that needs to be enforced on these actions, namely, *open_valve* $\succ$ *close_valve*. If a node sees these actions in the wrong order, it thinks that the valve is opened, while nodes seeing the actions in the causal order will treat the valve as closed. This may in turn lead to hazardous situations in the plant.

Thus the behavior $\mathcal{B}$ of a system interface may be prescribed in terms of the allowed temporal ordering relationships between various events possible at the interface level, given as: $\mathcal{B} \equiv (\succ, \{e\})$.

4.5 GUI for event specification

We have developed an interactive menu-driven window interface (GUI) between human users and the machines implementing the monitor system. The GUI allows users to inject a variety of critical scenarios in the problem-domain being modelled (e.g., 'battlefield' simulations).

The menus basically identify the set of system tasks as a set of 'buttons' displaying the verbose description of the physical events and state variables. When a user clicks on a button, a 'value' menu space pops open, indicating the possible values for the task type clicked. A set of applicative functions prescribed by the computation are also be loaded into the desktop as icons. Using these buttons and icons, a user can profile an event as satisfying any desired condition. Users can profile different system properties by clicking on these 'buttons' and invoking pre-loaded applicative functions in the desktop to operate on the variables. The level of user participation is left to his/her ingenuity and extent of computer knowledge.

Overall, we view our approach as *event filtering*, wherein an event filter is a programmable selection criterion for classifying or selecting events from a stream of events. The event filter is basically prescribed in a high-level declarative language that is compiled into the event monitor portion of the programming system. Since the events are *user-profilable*, our model of event monitor is applicable across a variety of problem-domains.

Figure 3 gives an architectural view of our monitor-based programming system.

5 Monitor API implementation

We have currently implemented the monitor software on 3 SUN Ultra-10 workstations interconnected by Ethernet, on top of HPOpenView software. The

[7]For instance, an underlying protocol mechanism may employ a periodic polling of sensors (based on authenticated sensor identification) to determine N_s and a profile comparison of sensor outputs in response to simulated test inputs to to the sensors to determine N_g. A call from the application (say, through a GUI) to monitor sensor reliability will activate the protocol mechanism to collect N_s and N_g, and then a synchronization algorithm that timestamps the collected data and orders them chronologically to determine if the event SENREL has occurred.

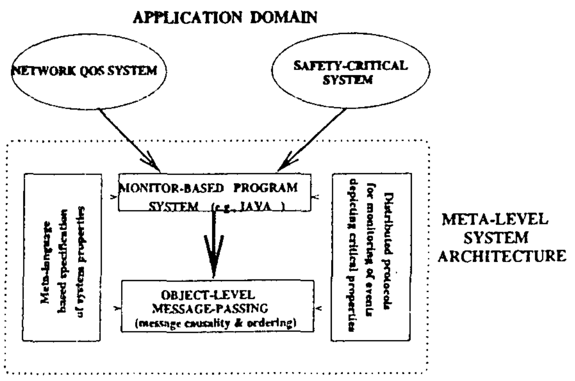

APPLICATION DOMAIN
NETWORK QOS SYSTEM
SAFETY-CRITICAL SYSTEM
MONITOR-BASED PROGRAM SYSTEM (e.g. JAVA)
OBJECT-LEVEL MESSAGE-PASSING (message causality & ordering)
Meta-language based specification of system properties
Distributed protocols for monitoring of events depicting critical properties
META-LEVEL SYSTEM ARCHITECTURE

Figure 3: Architecture of our programming system

monitor and computation modules typically co-reside in each workstation, interacting with one another through 'public class' routines. These routines are implemented through a JAVA API, providing access to event registry management and event detection mechanisms. JAVA Remote Method Invocation is employed for a distributed realization of these modules. The APIs exercise HP OpenView primitives to make use of the latter's built-in management and reporting capabilities [9], for implementing our model of event monitor.

The details of implementation are outside the scope of this paper.

6 Conclusions

The paper described a programming model for monitoring the service-level behavioral parameters that characterize the data flows in a distributed information system (DIS). We detect deviations from expected critical behaviors of a DIS as symptoms of information attacks (which may arise from component failures and/or external intrusion). Our new idea is the prescription of critical properties a DIS should satisfy as a set of logical formulae. These formulae can be transcribed into event predicates on the computation state, which can then be monitored by a distributed snapshot algorithm. The programming model comprises of a notation for specifying events and a composition mechanism for relating various parts of system interface state into events. The model exercises a decentralized structure of protocol agents at target computation nodes that monitor events during a program execution.

From a distributed programming perspective, our monitor-based structure of a DIS underscores two elements:

- Integration of the 'flow of real-time' into program-level notion of 'sensors' and 'actuators';
- Meta-level specification of events in terms of generic operators and applicative functions.

The meta-level specification method can be employed in a variety of application domains.

Overall, our programming model underscores a modular decomposition approach to monitoring distributed information systems, and hence makes their verification easier. This in turn reduces the cost of developing and maintaining dependable distributed systems.

References

[1] H. Kopetz and P. Verissmo. **Real Time Dependability Concepts.** Chap. 16, *Distributed Systems*, ed. S. Mullender, A-W Publ., 1993.

[2] E. Fabre, A. Benveniste, and C. Jard. **Distributed State Reconstruction for Discrete Event Systems.** In *Tech. Report* CDC00-REG1625, IRISA (France), March 2000.

[3] B. Dutertre and V. Stavridou. **Formal Requirements Analysis of an Avionics Control System.** In *IEEE Transactions on Software Engineering*, vol.23, no.5, pp.267-277, May 1997.

[4] P. Felber, R. Guerraoui and M. E. Fayad. **Putting OO Distributed Programming to Work.** In *Communications of the ACM*, pp.97-101, vol.42, no.11, Nov. 1999.

[5] S. S. Lam and A. U. Shankar. **Specifying Modules to Satisfy Interfaces: a State Transition System approach.** In *Distributed Computing*, Springer-Verlag, vol.6, 1992.

[6] R. Cooper and K. Marzullo. **Consistent Detection of Global Predicates.** In Proc. *Workshop on Parallel and Distributed Debugging*, ACM, pp.167–174, 1991.

[7] V. K. Garg. **Observation of Global Properties in Distributed Systems.** In Proc. *Intl. Conf. on Software and Knowledge Engineering*, IEEE CS, pp.418-425, Lake Tahoe (NV), 1996.

[8] K. Birman and T. A. Joseph. **Virtual Synchrony in Distributed Systems.** In Proc. *Symp. on Operating Systems Principles*, ACM-SIGOPS, 1987.

[9] HP WBM. **HP Proactive Networking: The Networking Management Component.** Hewlett-Packard white paper, 1998.

Proceedings of IEEE International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA 2005), Orlando (FL), June 2005. (published by IEEE Press)

Appendix 3

Performance Management Protocols for Adaptive 'content distribution networks' **

K. Ravindran & Jun Wu *

Abstract

The paper describes a management-oriented service model for cost-effective 'content access' provisioning to clientele. The service provider (SP) maintains multiple protocol mechanisms to replicate the 'content' pages at different proxy nodes of the distribution network. The mechanisms, which may embody server-driven and client-driven replica update schemes to synchronize the page copies, purport to reduce the access latency and data divergence on the 'content' pages retrieved by clients. Each protocol mechanism excels over the others in a distinct operating range of the distribution infrastructure. So, the SP may select an optimal protocol to meet the client-prescribed QOS obligations under the current levels of link delays and client traffic. We use the 'degree of proxy spread-out' as a protocol-level metric for the infrastructure resource usage – and hence the 'cost' incurred for content access. Our model allows 'dynamic switching' from one protocol to another as the infrastructure-level cost changes. The paper describes the management interface between the SP and network infrastructure to monitor resource usages and map them onto a specific level of proxy placement in the network. The management modules allow installing appropriate policy functions at proxy nodes to make the 'content access' provisioning resource-optimal.

* K. Ravindran and Jun Wu are with the Department of Computer Science, City University of New York (City College and Graduate Center), New York, NY 10031, USA — E-mail: *ravi@cs.ccny.cuny.edu*, *jwu@gc.cuny.edu*.

** Part of the work on service-level interface and distributed monitoring mechanisms was supported by AFRL-HE grant: F33615-03-1-6385 (2003).

1 Introduction

A 'content distribution' network (CDN) often employs an overlay of caching proxies placed at multiple nodes of a data network [1], to control the quality of service (QOS) of content access. With client access patterns changing over time and the information contents being dynamic, protocols that can be automatically reconfigured to realize efficient access to remote contents are necessary. Our goal is to incorporate ***policy-based control*** of the internal operations of a CDN based on various access performance factors. These factors include the location of clients in the network, their access patterns and the frequency of content changes.

With potentially a large number of clients trying to access various parts of an information (i.e., pages), the server maintains copies of various pages at multiple nodes of the distribution network [2]. When a client accesses the CDN server for a page, the request is forwarded to the nearest node containing this page for downloading. The client — which may be a web browser — uses the URL of a server page which is transparently mapped to the path leading to an appropriate node for downloading[1]. Keeping copies of the information base at multiple nodes in the network (i.e., replica nodes) increases access performance, particularly when the information accessed contains a large amounts of high bandwidth multimedia data. See Figure 1. AKAMAI [3] and Digital Island [4] are examples of commercially available CDN software systems.

To control the QOS of content delivery in highly dynamic settings, suitable measures are required to

[1]Consider an auto dealer who advertises the cars available in the lot for sale on a particular day through a CDN. The information about a car may itself be in the form of images of the car, video clips showing the driving condition of cars, and voice descriptions of the features and prices on the car. When a car shopper accesses the information about a car over, say, a web, all the annotative multimedia data for this car are presented to the shopper.

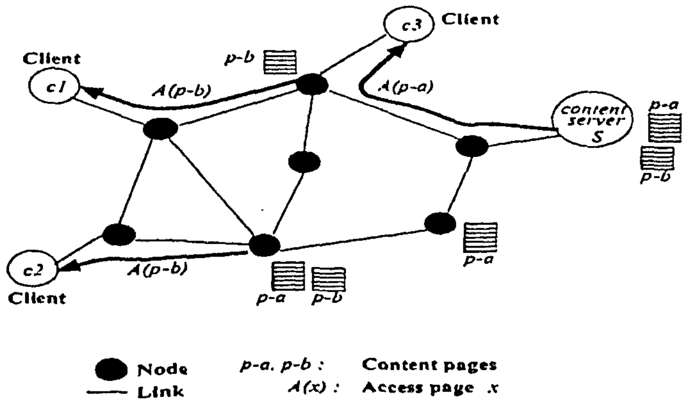


Figure 1: Structure of a CDN

evaluate the performance effectiveness of a CDN. From a client's perspective, the page *access latency* — i.e., the time delay between when a page is requested and when the page is made available — depicts a quality of the content delivery service. Another quality attribute is the page *divergence* — i.e., how current a page returned to the client is relative to most recent version of the page available at the server[2]. Both the QOS attributes are related to the page replication control exercised by the CDN. For instance, where the replicas are placed in the network (relative to client and server sites) and how often the replicas are updated (to keep them consistent), should be determined in a way to accommodate the latency and divergence needs of the application and to optimize the use of resources in the distribution network, i.e., communication bandwidth and node caches — refer to Figure 1. The policies for replica placement and page updates are chosen by the CDN service provider[3].

In general, different types of information contents may exhibit different latency and divergence characteristics. For example, stock market profiles may be posted on a NASDAQ web site once in every 10 minutes, with the market data presented to a client required to be no older than, say, 2 minutes; on the other hand, a weather forecast information may be posted on a Newsweb once in every 2 hours, with the weather data presented to a client required to be no older than, say, 1 hour. However, the content access protocol mechanisms for such diverse types of information can have a *generic* structure. Given this view, a CDN service provider may install generic access protocol mechanisms that can be instantiated with content-specific parameters. This allows a single set of protocol mechanisms to be re-used across different types of 'content access' provisioning (thereby lowering software development and maintenance efforts).

Our service model maps the internal mechanisms of a CDN to a *cost* based on the resources expended by them, namely, the amount of page replication and the message overhead for page synchronization. The model allows dynamic switching from one policy module to another, based on the cost associated with the infrastructure resource usage for a given level of access provisioning. The cost assignment for resource usage is prescribable in terms of the relative weights assigned to various resource parameters. Furthermore, the client entity may change the access QOS requirements mid-point based on its information needs, which may necessitate adjusting the underlying replication resource parameters. The overall goal is to monitor the access performance and adjust the replication parameters dynamically, to achieve a cost-optimal provisioning of 'content access' service.

We consider two alternate mechanisms for page updates in a CDN: deferring of page updates until a client request arrives (i.e., client-driven scheme CL) and propagation of updates every time the server changes its master copy (i.e., server-driven scheme SR). The CDN parameters that influence the choice of CL or SR schemes is the degree of replication (r) and the frequency of client requests relative to changes in the information content being accessed. There are inter-woven aspects to be considered: i) for what values of r the SR or CL scheme may be employed, and ii) what the cost and QOS considerations are that influence a switch from SR to CL scheme or vice versa. These aspects are specific to the policies on cost optimizations (and hence revenue accruals) from the CDN service offerings. Though the CL and SR schemes by themselves are not new, a management control of when to switch from CL to SR scheme, or vice versa, at run-time in a transparent and automatic manner (or, for that matter, between any set of update policies) is our innovative idea.

The paper provides the functional mechanisms to

[2]In the earlier example of accessing auto shopping information, as cars get sold to customers and new cars arrive at the lot, the car availability information gets changed. Furthermore, the price discount applicable on a car may also change, say, on an hourly basis. These changes should somehow be reflected on the car information accessed by customers to a reasonable level of accuracy.

[3]As an example of policy, a life-time may be associated for each page, upon the expiry of which a node hosting a copy of this page obtains a fresh copy from the server (say, in the earlier example of auto web, the price information on cars getting updated every hour). See [5] for a treatment of various replica update schemes for web content.

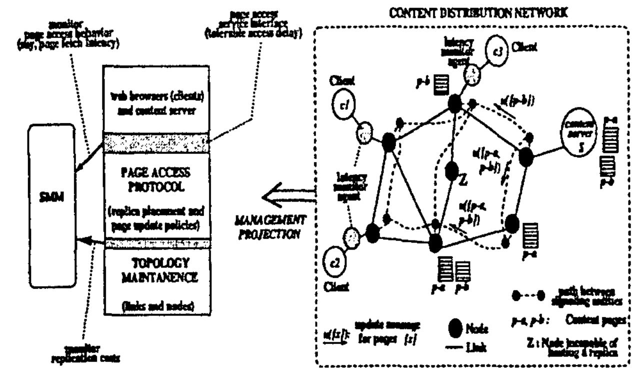


Figure 2: Service management view of a CDN

realize the switching from one policy function to another for a transparent service provisioning and to enable service-level reconfigurations to accommodate dynamically changing client needs. These mechanisms are based on our studies on different types replication based content access. The service model can be incorporated into a standardized middleware framework for web access (such as the application services architecture proposed in [6]).

2 QOS-based content access service

A 'content access' service is characterized by a set of QOS attributes, namely, the page access latency and the page divergence [2]. The client application prescribes these attributes when requesting 'content access' to the SP. How successfully the QOS attributes can be met is determined by the degree of replication maintained by the underlying access protocol and the replica update mechanisms employed therein. We describe such a QOS-controlled 'content access' service from a management perspective.

2.1 Service management-level control

In terms of our service model, the set of replica nodes that maintain copies of a page constitutes the 'infrastructure', and the update message overhead on these replicas constitute the 'resource'. See Figure 2 for an illustration. The protocol mechanism encapsulates a placement of replicas and an installation of update policy functions at various nodes of a CDN. A management module (SMM) maintains *agents* at the various sites of a CDN, to carry out service-level and/or service-internal reconfiguration tasks. The agents are supported through a signaling structure maintained by the SMM.

A proxy server entity resides at each replica node, encapsulating two functions: storing a copy of the 'content' and updating this copy. These functions are carried out by exchanging signaling messages between the server and proxies. Policy functions, such as 'server-driven' updates or 'client-driven' updates, may be installed at replica nodes through hooks from the management procedures in these nodes. The signaling is however oblivious of the type of policy functions installed. The policy functions may be dynamically dispatched by a management module (SMM) implemented by the CDN SP.

Another type of entities are placed at client sites to function as SMM agents. These agents continuously monitor the page access latency and divergence parameters so that clients can adapt their access behavior (such as removing a page from their access set if it incurs excessive latency). The type of adaptation can be prescribed through a set of policy functions compiled into the agent programs.

We[4] employ the JAVA programming language to realize the SMM functions, since JAVA allows dynamic 'function shipping' to remote sites.

2.2 Resource-based protocol selection

From a management perspective, the tradeoffs involved in choosing a protocol may be captured through a cost formulation that weights each of the tradeoffs with others on a normalized scale.

Let $R(q)$ represent the infrastructure resources expended by a service protocol corresponding to its internal parameters q — where $R(q) > 0$, and $\Theta_c(R(q))$ be a normalized cost, under an evaluation index c, associated with using the resources $R(q)$. The index c may depict the metric that is used to measure the performance — such as network usage, ISP's revenues, and data consistency (see [8] for a survey of performance metrics). The cost function satisfies the *monotonicity* condition, namely:

$$\Theta_c(R(q)) > \Theta_c(R(q')) \quad \text{or} \quad \Theta_c(R(q)) < \Theta_c(R(q'))$$

for $R(q) > R(q')$. The relation Θ is useful only as a measure of the cost variations with respect to changes in the underlying protocol parameter(s).

Consider our case of CDNs. Here, q may represent the placement of replica nodes in the network topol-

[4]Current works on quantifying infrastructure resource allocations (such as the results in [7]) can be used as a framework to determine policy functions.

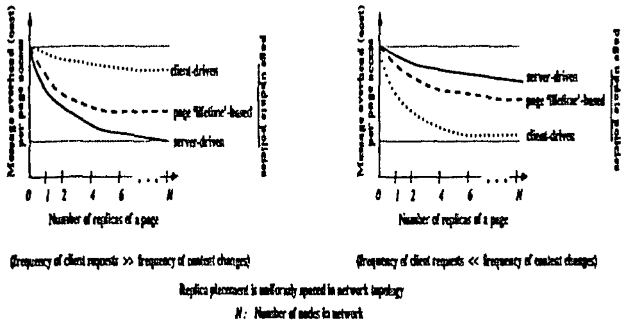


Figure 3: Page sync costs (an empirical study)

ogy (i.e., content caching nodes) and $R(q)$ may be the degree of page replication achieved using these replica nodes — denoted as *drep*. Likewise, if q represents the page update frequency, $R(q)$ may be the level of synchronization achieved among replicas — denoted as *sfreq*. Then[5] the average message overhead (normalized) in accessing a page may be given as:

$$\Theta_{msg}(drep) + \Theta_{msg}(sfreq).$$

Given that *drep* and *sfreq* are so chosen to keep the page fetch latency within the limits prescribed by clients, message overheads may be estimated for these chosen values of *drep* and *sfreq*. A comparison of message overheads under different parameter settings for the degree of replication and page update frequency may respectively be given by:

$$\Theta_{msg}(drep') < \Theta_{msg}(drep'') \quad \text{for} \quad drep' > drep''$$
$$\Theta_{msg}(sfreq') > \Theta_{msg}(sfreq'') \quad \text{for} \quad sfreq' > sfreq''.$$

With pre-defined policy functions to compute the cost components (in a normalized form) based on the protocol parameters *drep* and *sfreq*, the SMM may estimate the combined message overheads for the purpose of comparing different CDN protocols.

Figure 3 illustrates the normalized service-level cost variations with respect to the underlying protocol parameters. The cost illustration is based on empirical data collected from our studies of protocol mechanisms for CDN service offerings.

As can be seen, the cost model provides a *relativistic* measure of resource usage, rather than an absolute measure. Such a model suffices to compare and evaluate protocol implementations with different parameters and policies on a relative scale.

2.3 Monitoring of resource costs

The service management module SMM may sample the resource usage parameter $R(q)$ at chosen points in time, and then compute the service-level costs using the $\Theta_c(R(q))$ relation. Typically, the information changes over a slower time-scale, in comparison to the frequency of content access by clientele. This aspect may influence the choice of resource parameters.

In some cases, the parameters $R(q)$ can be computed in analytical form based on the samples of infrastructure parameters q collected by the SMM. An analytical estimate exemplifies a static mapping of infrastructure parameters q to protocol-level resource parameters $R(q)$. This is possibly a case where 'lifetime' based page updates is employed by the CDN protocol.

In some other cases, the parameters q may not be available for external sampling, in which case, $R(q)$ itself may need to be sampled by auxiliary instrumentation deployed at service-level. These cases arise when the representation of protocol internal resource parameters is not expressible in a analytic form that can be transcribed into a programming code. This is possibly a case where 'client-driven' or 'server-driven' updates are employed. Here, the CDN SP needs to implant on-line probes into the access protocol layer to measure the resource parameters over problem-specific time scales. In general, a measurement-based probe may be viewed as a 'black box' that estimates the required resource parameters.

Regardless of the underlying mechanism for estimating the resource usage parameter $R(q)$, the SMM may determine how the service-level costs $\Theta_c(R(q))$ change over time in using various alternate protocols. Typically, a sustained higher cost of of the protocol actually employed for providing a service S relative to other competing protocols can trigger the selection of an alternate protocol for S. The decision as to when a protocol switch should be undertaken may be based on a policy function that interprets the cost variations.

2.4 server-driven vs client-driven update

We consider two alternate mechanisms for page updates in a CDN: deferring of page updates until a client request arrives (i.e., client-driven scheme CL) and propagation of updates every time the server changes its master copy (i.e., server-driven scheme

[5]With client-driven updates of page copies, *sfreq* may be the frequency of client requests for page access; with server-driven updates, *sfreq* may be the frequency of updates occurring on server-maintained pages; with 'life-time' based page replenishment, *sfreq* may be determined by the life-time associated with pages.

SR). We discuss below how to switch from one mechanism to another at run-time — c.f. section 2.2 for the underlying cost tradeoffs.

The CL scheme requires that each copy of a page carry the time-stamp of its last update (LTS) and the most recent update on this page that has occurred across the entire network (GTS). Suppose a client request arrives at a proxy node X for a page p. If $\mathrm{LTS}(X,p) = \mathrm{GTS}(X,p)$, X sends its copy of p to the client. If $\mathrm{LTS}(X,p) < \mathrm{GTS}(X,p)$ — meaning that X's copy of p is obsolete, X contacts another proxy node Y that has $\mathrm{LTS}(Y,p) = \mathrm{GTS}(Y,p)$, and then obtains the copy of p from Y for forwarding to the client. When the content server Z updates its master copy of p, it sets $\mathrm{GTS}(Z,p)$ to the current time, and then propagates this time-stamp to all the proxy nodes for p for updating their LTS values. In the SR scheme, every proxy node has the most recent copy of p, i.e., the copy at each proxy node is the same as that at Z. This is achieved by having Z propagate an update to all the replicas whenever the master copy of p at Z changes.

The SMM-user signaling and SMM-network signaling allows instantiating a generic SP-internal function that allows switching between SR and CL modes and a service-level reconfiguration function with the SP-specific parameters and client-level parameters.

3 Switching between update schemes

We provide a management-level description of the (meta-)activities involved in dynamic switching of the update schemes.

Consider[6] a switch in the protocol mechanism, say, from CL to SR. Since the SR scheme does not support time-stamps, new client accesses should be blocked and any in-progress client accesses under the current CL scheme should be completed, before the SMM initiates the switching. When there is no in-progress client access, the content server may initialize all the replica nodes with its most recent version, and then unblock any pending client accesses for processing under the SR scheme. The switching is illustrated by the following code skeleton:

```
protocol_switch (CL, SR)()
    suspend new client invocations for page access;
    wait until in-progress client invocations
            have completed under CL scheme;
    ∀x ∈ REPLICA   /* REPLICA:  set of replica nodes */
        ∀p ∈ PAGE   /* PAGE:  set of pages */
            if (LTS(x,p) < GTS(x,p))
                propagate update of p
                        from content server to x;
    resume client invocations
            for page access under SR scheme.
```

[6]The updates of GTS(_, p) at various replicas in the CL scheme and the updates of all copies of p in the SR scheme are assumed to be atomic relative to the occurrence of client requests.

For a switch from SR to CL however, the content server simply needs to assign the current time as the GTS and LTS for every page copy maintained by various replica nodes, as given by:

$$[\forall p \in \mathrm{PAGE}\ \forall x \in \mathrm{REPLICA}$$
$$\mathrm{GTS}(x,p) := \mathrm{LTS}(x,p) := \text{current time}],$$

and then start processing client invocations for page access under the CL scheme.

The resource-level parameter that influences the choice of CL or SR protocols is the degree of replication (r) and the frequency of client requests relative to the time-scale of changes in the information content being accessed. There are inter-woven aspects the SMM needs to consider:

1. Under what degree of replication the SR or CL protocol may be employed;
2. What the cost and QOS considerations are that influence a switch between SR and CL modes.

These aspects are specific to the 'content access' SP policies on revenue accruals from its service offerings. Figure 4 shows a profile of the protocol choices made at various points in time for a sample scenario involving the changes in r and client-prescribed QOS.

The discussions so far assumed that the user-level quality prescription remains fixed. We now describe the SMM-to-user interface procedures, to facilitate dynamic changes in client-prescribed quality.

4 Client-level reconfigurations

We take the infrastructure resource parameter that impacts the behavioral attributes of the 'content access' service visible to clients, and then describe the meta-activities carried out by SMM to facilitate the adjustment of these parameters at run-time. Figure 5 illustrates the service-level behavioral variations with respect to the underlying infrastructure resource parameter, namely, 'degree of replication'. This behavior

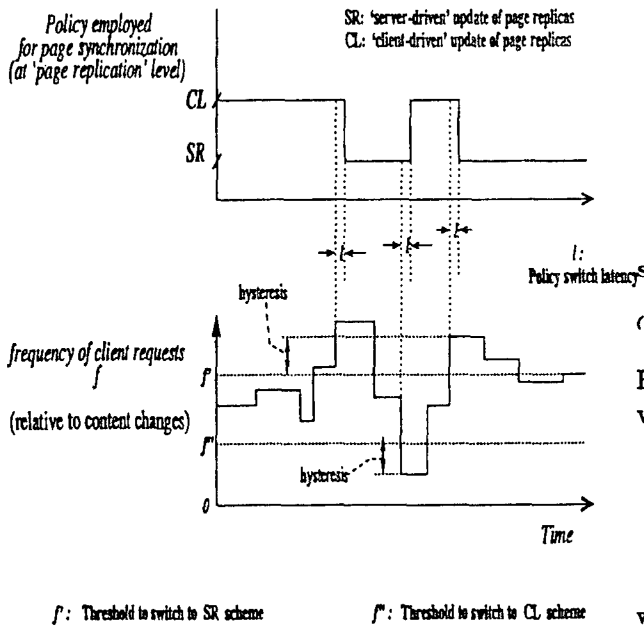


Figure 4: Time-scales of parameters variations at different layers of a CDN

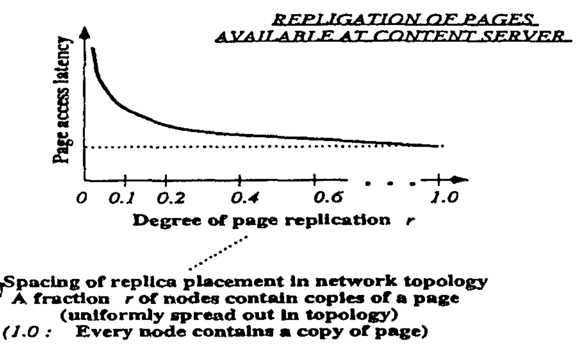


Figure 5: Behavior-oriented study of CDN services

can be monitored by directly sampling the interface state of 'content access' services. The behavior illustration is based on empirical data collected from our separate studies of the CDN protocol mechanisms.

The page access latency PLAT, which is a client-visible attribute, is influenced by the replication mechanism that allows the nearest and most-recent copy of a page in the network to be accessed — refer to Figure 5. The protocol-level mechanism depicts the exercising of infrastructure resources, namely, 'sync' message exchanges and traversal of links in the network.

The internal state of a protocol that realizes the CDN service is the current placement of replicas $repset_{cur}$ and the link delays between nodes $ldel_{cur}$. The SP may provide a mapping function:

$$plat(i) = \gamma_{(\mathrm{CDN},U)}(repset_{cur}, ldel_{cur}),$$

to yield the value of PLAT in an i^{th} sampling interval, where $\gamma_{...}$ depicts a static one-to-one mapping function. The function may be specific to an update policy U on pages employed by the CDN protocol (i.e., client-driven or server-driven or page lifetime-based).

We consider four types of operations provided by the CDN SP for use by the SMM to control the replica placement:

$plat(obs) = \mathrm{smooth\_latency\_samples}(\{plat(i)\}_{i=1,2,\cdots});$
$plat_{new} = set\_latency(plat'_c, plat(obs));$
$repset_{new} = \mathrm{get\_replica\_placement}(plat(obs), plat_{new});$
$\mathrm{change\_replica\_placement}(repset_{new});$

where $\{plat(1), plat(2), \cdots\}$ are the time-chronological sequence of samples of PLAT from which a smoothed observation $plat(obs)$ can be extracted and $repset_{new}$ depicts a new placement of replicas on the CDN nodes to bound PLAT to less than $plat_{new}$. The 'smooth_latency_samples' operation may simply be an averaging mechanism over a time-scale meaningful to the 'content distribution' application. The condition $> (plat(obs), plat_c)$, depicting that the monitored value $plat_{obs}$ exceeds $plat_c$, will cause the SMM to trigger a change in replica placements (say), where $plat_c$ is the currently prescribed value for PLAT.

The 'set_latency' operation may compute $plat_{new}$ as, say, $[plat'_c - \Delta \times (plat(obs) - plat'_c))]$, where $plat'_c$ is a modified limit on latency prescribed by the client and Δ is a parameter used by the SP for 'control-theoretic' stability of the changes in latency. The 'get_replica_placement' function may employ the $\gamma_{(\mathrm{CDN},U)}$ function in determining $repset_{new}$ that will cause the latency to not exceed $plat_{new}$. The 'change_replica_placement' function is specific to the update policy employed by the CDN protocol. Suppose a server-driven update policy is employed. If $repset_{new} \supset repset_{cur}$, then the replicas $(repset_{new} - repset_{cur})$ are initialized to the current contents of the page, and then installed at the designated CDN nodes. If $repset_{new} \subset repset_{cur}$, then the replicas $(repset_{cur} - repset_{new})$ are removed from the concerned CDN nodes. Other cases can be a combination of these mechanisms.

The operations listed for the CDN services are supplied by the SP as 'applicative' functions that can be invoked by the SMM. The mapping functions may depict infrastructure resource control policies implemented by the SP (such as server-driven replica updates). The time-scales associated with the changes in behavioral parameters are also prescribed by the SPs. The client prescribes the monitoring conditions through 'applicative' functions: the '>' relation on access latency values, for use by the SMM[7].

5 SMM prototyping considerations

Diverse applications may exist, requiring the delivery of different types of information contents. However, their behavioral characteristics at service interface level and how these characteristics interplay with infrastructure resources exhibit a generic structure. For instance, the relationship between the client-visible attributes: access latency (PLAT) and page divergence (PDIV), and the resource-level parameters, namely, degree of page replication (r) and synchronization message exchanges ($sfreq$), can be determined either by off-line empirical studies on CDNs and/or by closed-form analytical expressions[8] [9, 10]. So reducing the software development and maintenance efforts for CDN management requires implementing the SMM as a set of generic functions for replication control at the CDN infrastructure level and then instantiating these functions with the client-specified PLAT and PDIV parameters. See Figure 6.

5.1 Sub-system structures of SMM

There are three sub-systems to be considered in prototyping the SMM:

1. A sub-system implementing the CDN functionality that is reconfigurable at service interface level (SINT);
2. A sub-system that binds the infrastructure-level resources to the interface-level QOS parameters (SRES);
3. A sub-system that monitors the CDN behavioral parameters at service interface level and resource parameters at infrastructure level (SMON).

[7][6] provides architectural frameworks to encapsulate policy functions in systems for managing service-level infrastructures.

[8]There may be other parameters as well which influence PLAT and PDIV (e.g., the dependence of PLAT on page sizes). To keep our discussion focused on the SMM, we assume — without loss of generality — that these parameters do not change during a CDN service provisioning.

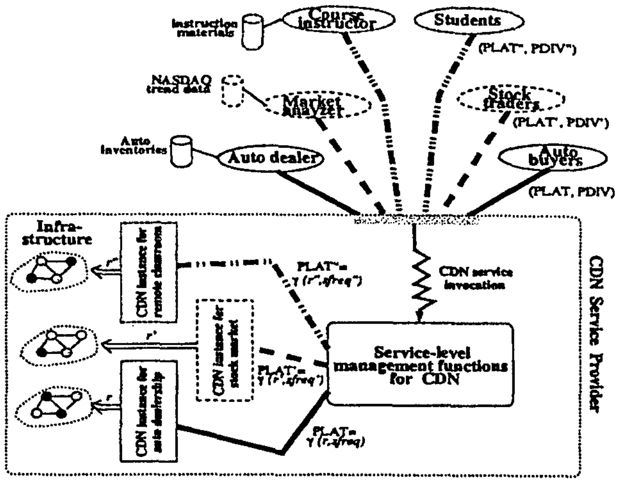


Figure 6: SMM-based CDN service offerings in diverse applications

These sub-systems are not specific to a particular type of information content being delivered (i.e., are application-independent).

Interface event prescriptions may be based on logical formulas relating the client-visible interface state variables that are loaded into the SINT function. For instance, '$\mathrm{PLAT} < \delta \wedge \mathrm{PDIV} < \zeta$' may capture the cases of the observed latency PLAT and/or divergence PDIV exceeding the client-prescribed limits δ and ζ respectively. These formulas, when evaluating to false at run-time, may be indicative of QOS violations. Cost relations on infrastructure resources, i.e., $\Theta_{msg}(r)$, may be loaded into the SRES function for dynamic evaluation of the cost-effectiveness of a given replication control mechanism. The SMON function monitors the PLAT and PDIV parameters, and also the degree of replication r. The time-scales of variations of the interface and resource parameters are loaded into the SMON function for a run-time instantiation of the monitoring component.

5.2 Registry based 3-way handshake

The binding information for a CDN service is maintained by the SMM in a registry, as a parameter template prescribed by the SP. The binding information includes 'event' prescriptions, using logical relations $L(par)$, that symptomize QOS requirements. When a client invocation is received by the SMM, it creates a new instance of SINT and SRES, and invoke them with an appropriate mapping relation γ and a replica control mechanism (such as SR or CL) respectively. The cost relation Θ_c and event condition $L(par)$ are loaded into the SMON for monitoring purposes.

In our study, we realize the SINT and SRES functions as black-box systems that provide external behaviors close to that of actual CDNs. These equivalent systems maintain pre-computed behavioral profiles: both at service interface level and infrastructure level, that are exercised by randomly generated stimuli. The profiles are stored in the form of tables relating the various parameters. The time-scale information is also loaded into the black-box functions as the 'time-constants' of first-order and second-order representation of behaviors. For instance, the stimulus may be a change in PLAT parameter, whereupon the SINT function determines the required level of replication r — i.e., PLAT = $\gamma(r)$, whereupon the SRES function adjusts the page replication based on r, and also, changes the replica update mechanism if necessary.

We believe that studying a CDN behavior through an 'equivalent' system that generates a behavior close to the actuals (in a mathematical sense) suffices for our goals of studying the SMM functionality.

6 Conclusions

We have described a management model to enable a cost-effective provisioning of 'content access' services for QOS-adaptive applications. The model can be employed by content providers to optimize the usage of network infrastructure resources in dynamic settings.

The 'content access' service provider (SP) maintains multiple protocol mechanisms to replicate the content pages at different nodes of the distribution network. The mechanisms, which may embody server-driven and client-driven replica update schemes to synchronize the replicated pages, purport to reduce the page access latency and the level of data divergence on the content retrieved by clientele. Client applications prescribe the required QOS obligations, with the SP instantiating one of the protocol modules to meet the QOS parameters. The management module (SMM) may dynamically switch from one protocol module to another, based on a notion of cost associated with the infrastructure resource usage, namely, the degree of content replication, for a given level of QOS offering. The SMM tracks the changes and/or outages in network infrastructure resources in a dynamic setting, and map them onto a cost of 'content access' provisioning through the selected protocol.

The paper described the service-level interface between the SMM and network infrastructure to monitor resource usages and between the SMM and client entities to monitor QOS offerings. The management-level control enables installing appropriate policy functions that can make the 'content access' provisioning resource-optimal. The underlying signaling primitives enable the SMM to carry out transparent protocol switching and/or trigger client-level reconfigurations.

Since the SMM role is generic, multiple instances of CDN service provisioning can be deployed to cater to different applications.

References

[1] J. W. K. Hong, S. Heilbronner and R. Wies. **Web-based Intranet Services and Network Management.** In *IEEE Communications*, Oct. 1997.

[2] J. Chase, S. Gadde and M. Rabinovich. **Web Caching and Content Distribution: a View from the Interior.** In 5th Intl. Workshop on *Web Caching and Content Delivery*, Lisboa (Portugal), 2000.

[3] Akamai Technologies, Inc. **FreeFlow Content Distribution Service.** In URL: *http://www.akamai.com.*

[4] Digital Island, Inc. **Footprint Content Distribution Service.** In URL: *http://www.digitalisland.net/services/footprint.shtml.*

[5] P. Rodriguez and S. Sibal. **SPREAD: Scalable Platform for Reliable and Efficient Automated Distribution.** In *Computer Networks* (North-Holland publ.), vol.33, no.1-6, pp.33-49, 2000.

[6] G. Kar and A. Keller. **An Architecture for Managing Application Services over Global Networks.** In proc. *INFOCOM'01*, IEEE-CS, Anchorage (AK), April 2001.

[7] H. M. Mason and H. R. Varian. **Pricing Congestible Network Resources.** In *IEEE Journal on Selected Areas in Communications*, vol.13, no.7, pp.1141-1149, Sept. 1995.

[8] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. van Steen. **Replication for Web Hosting Systems.** In *ACM Computing Surveys*, 36(3), Sept.2004.

[9] L. Qiu, V. N. Padmanabhan and G. M. Voelker. **On the Placement of Web Server Replicas.** In Proc. *IEEE INFOCOM'01*, Anchorage (AK), April 2001.

[10] E. Cohen and H. Kaplan. **Refreshment Policies for Web Content Caches.** In Proc. *IEEE INFOCOM'01*, Anchorage (AK), April 2001.

Proceedings of 9th IEEE International Symposium on Distributed Simulation and Real-time Applications (DS-RT 2005), Montreal (Canada), pp. 236-245, October 2005.

Appendix 4

Event-based Programming Models for Monitoring of Distributed Information Systems

K. Ravindran & Jun Wu *
Department of Computer Science
City University of New York, USA.
Contact e-mail address: *ravi@cs.ccny.cuny.edu*

Abstract

The paper deals with providing a specification methodology to enable flexible monitoring in large scale distributed information systems (DIS). The goal is to allow the monitoring of compliance to properties deemed as critical for functioning of a DIS. Since information attacks due to external intrusion and/or component failures often manifest as deviations from critical behaviors expected of a DIS, a violation of compliance to a specified critical property can be viewed as symptomatic of information attacks. Based on this principle, users can prescribe critical properties in the form of *event predicates*, which are boolean conditions on the externally visible interface state distributed among computation nodes and can be detected by distributed algorithms. Our specification methodology manifests in two facets: i) designing a meta-language which allows the interface behavior of a system to be prescribed through possible occurrences of events in the external environment; and ii) incorporating the specification language into a programming interface that exposes the computation-level states symbolizing the occurrence of events. Any target application can then plugged-in to the generic monitor tool to test for its compliance to critical functionality requirements. The realization of monitor in distributed object-oriented programming languages (such as JAVA) reduces the software development costs of implementing a DIS: due to the ease of specification, development and maintenance of the DIS.

* Part of the work on user-interface support for distributed monitoring is currently supported by AFRL-HE grant: F33615-03-1-6385 (2003). Part of the work on programming and algorithmic aspects was earlier supported by PSC-CUNY Award (2000).

1 Introduction

Large scale applications in the future require a control paradigm that is based on 'information intensive' computations where vast amounts of data need to be processed by decision-making entities to carry out strategic operations in real-time. Examples of newer information networks are networked-sensor systems for terrain surveillance in geological settings and for weather forecasting in atmospheric settings (say, to predict earthquakes and storm conditions respectively). These applications are 'born to distribution' in that their very conception is based on processing information placed in different computation nodes (or subsystems). The computation nodes are interconnected by an information distribution network that makes the relevant pieces of data available to them at various points in time, to enable them carry out their assigned task [1].

The strengths arising from the very nature of decentralized processing and control in a distributed information system (DIS) — such as increased tolerance to independently occurring component failures — also bring in some problems along with. A main problem is that a piece of distributed information is more vulnerable to *attacks* due to uncontrolled changes induced by 'ripple-effect' failures, such as a damaged sensor skewing the reading of physical parameters in an undetectable way. Where a DIS handles highly sensitive data (such as in 'digital battlefield' settings and in radar tracking of airspaces), information attacks can also occur due to planned intrusions by mischievous outsiders. Regardless of the physical manifestation of an information attack, the integrity of various information pieces can get compromised, i.e., a piece of information may assume a value outside a given range that may be indicative of a normal operation of the

DIS. This[1] information-level failure can in turn lead to an anomalous behavior and/or a degraded behavior of the overall system.

Though attack watchdogs can be instituted at various sensitive points of a DIS with the hope of preventing an information from being compromised in the first place (e.g., the use of stand-bys for primary radar units in an airspace tracking system), the very unpredictable nature of an attack precludes the design of a comprehensive prevention mechanism that can catch all possible types attacks before their effects can penetrate into the system. When information failures do occur (despite institution of some form of prevention mechanism), how does a DIS cope with the resulting system-level anomalies ? This question motivates the thrust of our paper.

The goal of our paper is to provide a specification methodology that allows the monitoring of system-level properties deemed as critical for functioning of a DIS. Since an information attack often manifests as a deviation from critical behaviors expected of a DIS, a violation of compliance to a specified critical property can be viewed as symptomatic of information attacks. Based on this principle, users can prescribe critical properties in the form of *event predicates*, which are boolean conditions on the externally visible interface state distributed among computation nodes and can be detected by distributed algorithms. The event predicates include a prescription of the 'flow of real-time' as meaningful in the problem-domain.

From a programming perspective, our specification methodology consists of two facets:

1. A meta-language which allows the interface behavior of a DIS to be prescribed through possible occurrences of events in the external environment of the application;

2. Incorporating the specification language in a reactive programming system that exposes the computation-level states symbolizing the occurrence of events.

These facets can be built into a generic monitor API that can be instantiated with problem-specific parameters and functions. Any target application can then be plugged-in to the monitor tool to test for compliance to its critical functionality requirements. A system developer can easily augment the monitor API with application-specific libraries to form a complete distributed programming system.

[1]We use the term 'information attack' to refer to physical and/or logical manifestations in one or more system elements that perturb their visible behaviors to outside the normal operating points of these elements.

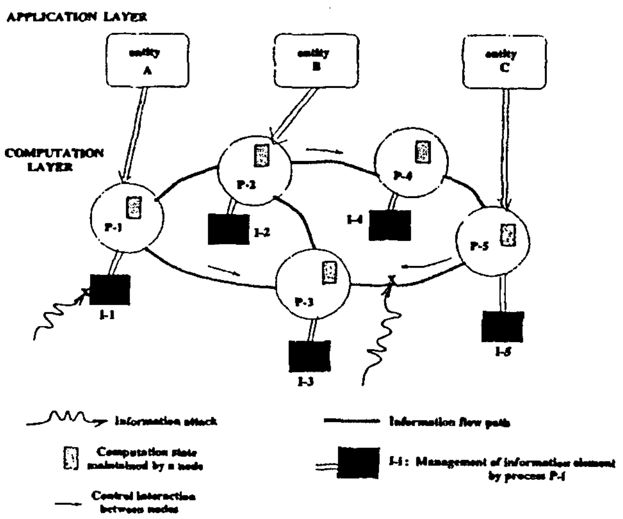


Figure 1: A bird's view of a distributed information system (DIS)

The realization of a generic monitor in distributed object-oriented programming languages (such as JAVA) can reduce the software development costs of implementing a DIS: due to the ease of specification, development and maintenance of the DIS.

The paper first begins with the generalized structure of a DIS to expose the underlying system-level issues. It then narrows down to our proposed specification methods for system monitoring.

2 Behavioral view of Distributed information systems (DIS)

An information system for large scale real-time applications may be viewed as carrying out a distributed computation that interacts with external environments consisting of:

- Data sources that function as *sensors* to track the changes occurring in the application environment (e.g., ground vibration analyzers to detect seismic variations);
- Data sinks that function as *actuators* to map computation results onto specific effects in the application enviroment (e.g., a motor to position a radar unit).

Human users and/or intelligent autonomous agents at distinct geographic locations may be part of the DIS, interacting with one or more computation nodes to provide strategic decision support. See Figure 1.

2.1 Why monitor system behaviors ??

The behavior of a DIS as a whole may be continually changing, sometimes to even below acceptable threshold levels of operations. This may be due to sub-system level components (viz., computation nodes and/or their interconnections) being forced to operate in a degraded mode because of intrusion by attackers, component failures and/or undesirable changes occurring in the external environment. Visible changes in a system behavior mean that the behavior is measurable in terms of concrete parameters of the system.

Consider, as an example, sensor devices deployed in battlefield terrain, say, to detect enemy movements. These sensors are prone to failures (such as loss of battery power to sensors). At a macro level, the reliability of inferences made about enemy movements, which depends on the accuracy and timeliness of sensor data, is a measurable property of the detection system. Some form of 'majority voting' technique (such as 'at least 10 sensors in the terrain have detected enemy movements) may be employed to increase the quality of inferences. The decision-making by commanders (say, to initiate attacks on enemy positions) may itself may be based on such a quantitative assessment of the 'inference reliability' (or confidence level) parameter[2].

Consider, as another example, a consensus protocol that reaches (approximate) agreement [2] on the temperature data reported by multiple sensors in an automated chemical plant. A damaged sensor may read a temperature value in such a way that it is within the acceptable range of values but is sufficiently different from the correct reading to skew the final result (to be agreed upon among sensors) in such a way to mis-direct any temperature control actions.

With such failures, it may often be the case that a failure can be detected after some damage has been inflicted upon the information. Since a contaminated information affects the visible behavior of a DIS in some way, we need to instrument mechanisms for detecting deviations from the expected behaviors. The issue then is on sufficiently early detection to prevent any further damage.

[2]Changes occurring at system hardware and/or software level may be abstracted into changes occurring at the information level. In the example of sensors, the effect of battery power-down may be subsumed into an observable reduction in the confidence level on 'enemy movement' information derived from sensor data. So a DIS should be modularized in such a way that macro-level anomalies of interest can be inferred from the easily observable behavior of system components.

2.2 Critical system properties

The acceptability of a system may be based on whether its observed behavior meets certain criteria that are absolutely essential in order for it to function without interruption. A prescription of these essential criteria may be deemed as a *critical property* of the system. A critical property may be associated with specific *symptoms* that can be observed by a management module in the computation subsystem.

Suppose, in the earlier example of sensors in a 'digital battlefield' system, it is stipulated that the confidence level in the reliability of 'enemy movement' information exceed a certain threshold, say 70%, in order to initiate a pre-emptive strike on enemy positions. When the number of active sensors falls below the corresponding 'majority' figure in the number of sensors detecting enemy movement (say, 70% of the sensors), the detection system can no longer be able to supply the information at the critical level of reliability.

In a broad sense, critical properties may cover many aspects of system behaviors pertaining to quality of information delivery (QOS) to the application. Here, quality is measurable through macroscopic indices capturing the timeliness, accuracy and/or precision of the information presented. Some instances where information quality can suffer due to incursions from the external environment are:

- Transport quality offering in data network settings (e.g., an increase in network delay jitter causing frequent glitches in a remote video presentation);
- Faults occurring in system components (e.g., the crash of a replica node that reduces the 'data availability' in an information repository);
- Malfunctions in subsystem-level components and/or their interconnections (e.g., failure of communication links in a distribution network that reduces the connectivity among nodes).

The above categorization is not exhaustive though. It is up to the application to determine what constitutes a mandatory property that must be met during a computation.

Regardless of the criteria in enunciating a critical system property, it is necessary to continually monitor a system behavior, so that reasoning about when the system meets its critical expectations and when it fails to meet can be made. Due to the real-time nature of information, detection of critical property violations that may arise from information attacks is also time-sensitive.

2.3 A real-time programming view of 'information failures'

Critical property violations are often symptomatic of information failures. Monitoring of system behaviors to detect violations is different from a recovery that often ensues when a violation does occur. Here are some key aspects to be considered in designing a monitor.

'event-action' based programming framework

The behavioral symptoms of a DIS should be programming-level projections of the anomalies occurring in the problem-domain (i.e., deviations from expected critical behavior) that can be observed across the management interface between decision-making entities in an application and the underlying computation sub-systems. The management interface should be programmed to notify the application entities when the symptomatic parameters fall outside the acceptable range of values. In the example of sensor failures, the 'inference reliability' parameter may be continuously tracked on a management console through a GUI, with the GUI raising, say, an audio-visual alarm when the parameter falls below 70%. Such a'call-back' notification may trigger a recovery action by the application entities (e.g., the commander at the console replacing the faulty sensor banks with good ones, or resorting to alternate means of inferring enemy movements).

Faster time-scales of monitoring

The parameters monitored by the management module should vary at much smaller time-scales than the anomalies in the problem-domain they symptomize. This is because the problem-domain has a much slower dynamics in comparison to the dynamics of the program-level parameters chosen to map to the problem-domain. In the earlier example of sensor failures, changes in the 'inference reliability' parameter can usually be detected in the time-scale of running a distributed algorithm on computation nodes that conducts the voting among sensors, which is much smaller than the time-scale of the effects sensor failures have on the 'battlefield' operations (such as unnoticed enemy movments). The[3] information dissemination algorithms implemented by application entities typically interpret the monitored program-level parameters at their native time-scales, in order to recover from, say, a mal-function in the problem-domain in a timely manner.

The above requirements of system-level monitoring should be reflected in the specification methods that describe system-level critical behaviors.

3 Reactive programming based approach

We break up a DIS into a physical system that is being controlled, and a set of sensors and actuators through which the physical system 'connects' with a computation sub-system. The latter consists of an *information collector* that pre-processes the 'sensed' data from the physical system, a *distributed monitor* that examines the sensor data to ascertain the system compliance to prescribed critical properties, and a *repair controller* that generates recovery actions in the form of driving the actuators to perturb the physical system in a way to ameliorate any deviation from expected critical behavior. The overall goal of information processing algorithms implemented in the computation subsystems is to keep the operating point of the DIS at a desired level. See Figure 2. As can be seen, the 'sensors' and 'actuators' may be abstract in nature delegated with a computational intelligence to carry out pre-processing and/or transformations on the input data and output results.

Since a bulk of the monitoring functionality is generic in nature, a general purpose monitor module that can be instantiated with problem-specfic parameters, functions and time-scales. Our proposed approach is based on this premise.

3.1 Limitations in current programming models

Currently available programming models for 'classical' real-time control (as in avionics and industrial applications) often employ an *integrated* approach, where the monitoring and control functionality are blended

[3]Consider, as another example, a video distribution application. A degradation in the quality of presentation due to a reduction in video frame rate is perceived by receivers much slowly, in comparison to the time-scale over which the frame rates actually fluctuate (say, due congestion in the video transport network).

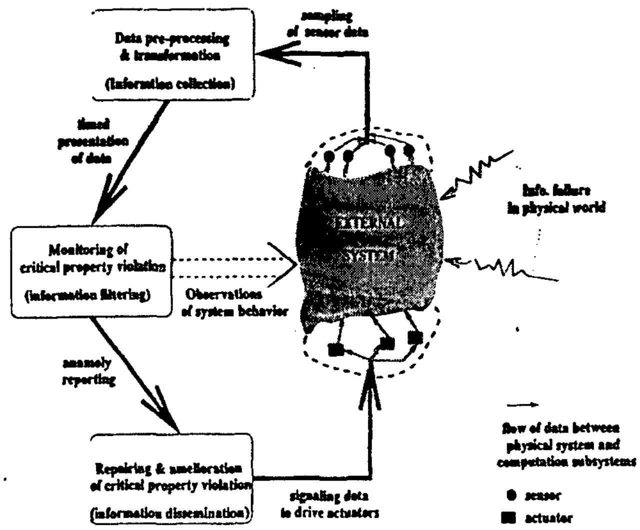

sampling of sensor data
Data pre-processing & transformation
(information collection)
timed presentation of data
Monitoring of critical property violation
(information filtering)
Observations of system behavior
EXTERNAL SYSTEM
Info. failure in physical world
anamoly reporting
Repairing & amelioration of critical property violation
(information dissemination)
signaling data to drive actuators
flow of data between physical system and computation subsystems
● sensor
■ actuator

Figure 2: Monitor-based structuring of a DIS

into the computation modules that constitute the target system being controlled [3]. The monolithic structure of computations underlying an 'integrated' programming view does not allow an easier separation of the application-independent and application-specific processing activities across the (management) interface between computations and external environment. Consequently, any behavioral description of a DIS needs to be hardwired into the computational procedures realizing the system. Such behavioral descriptions cannot be extracted out from the overall system specification for the purpose of explicit representation and manipulation by general purpose modules. This makes it difficult to re-use the functionality built into one system for another system, which manifests as redundancies in distributed software development.

Thus, we need a *modular* approach whereby the computational views of the physical system structure and the temporal behaviors of the system can be captured through generic programming modules instantiated with problem-specific parameters.

3.2 Monitor-based structure of DIS

Monitoring is a management function that co-exists with the target system itself, with the intent of controlling the system behavior. Since this management function is separate from the problem-domain functions implemented by the computation modules and recovery modules, it is necessary that the system requirements are prescribed in an application-independent form,, say, *declaratively* using a meta-language.

A variety of distributed application systems can be viewed through the monitor-based programming structure. For this purpose, a system may be modularized by prescribing two aspects:

1. The sensor and actuator modules that abstract the computational interfaces to the target physical system being controlled (i.e., the external environment of computation);
2. The time-scales of information flows between the computation modules and the sensor and actuator modules.

With the above aspects captured using a meta-langauge, the monitor module and the repair controller module can be instantiated with problem-specific parameters and 'applicative' functions to manipulate the sensor data and actuator drive signals through a generic schema.

Essentially, the monitor-based structure treats the 'sensors' and 'actuators' as encapsulated computational objects and the computation sub-system as an I/O automaton interfacing with these objects. Data from the 'actuators' cause state transitions in the computation that symbolize changes in the physical system. The 'sensors' track the state changes occurring in the physical system, with the monitor serving as a filter to detect only the meaningful changes that need to be notified to the 'repair' module. Such a computational view allows incorporating *dynamic adaptivity* in the functionality offered by a DIS[4].

4 Specifying critical properties of DIS

From a programming perspective, critical properties constitute invariants of a system. Occurrences of specific system-level events may constitute *symptoms* of situations where critical properties are violated. So system monitoring basically amounts to detecting the occurrences of specific events associated with these properties[5].

[4]From a Software Engineering perspective, the monitor-based programming view of a DIS opens up a new door for structuring the various modules of a complex embedded system, making it easier to understand the interactions between modules, enabling incremental development of system functionalities, and eliminating redundancies in the development of distributed software.

[5]'event detection' is a first-class abstraction, the complexity of which is encapsulated behind well-defined proce-

4.1 Temporal dimension of information elements

An information element $I_i|_{i=1,2,\cdots}$ maintained by a DIS may be given by a pair: $(I_i, val_i(t))$, where $val_i(t)$ is the value assumed by the element I_i at time t. Because of changes occurring in the external environment, the information elements are in a perpetual mode of changing (over time). Certain changes in val_i's may indicate a significant deviation in the external environment relative to its current operating point, which may be deemed as an 'event' [5].

We may prescribe a time scale ϵ_i over which changes in I_i occur, i.e., ϵ_i may be viewed as an average time interval between changes in I_i. For example, a noticeable change in aircraft coordinates provided by a tracking radar may not occur any more frequent than once in 5 *sec*. We refer to this property as 'element I is ϵ-stable'. The ϵ parameter may represent the time interval for sampling the sensor data emanating from external environment.

Suppose an event pertaining to I_i is observed to occur at time t. An event handler starts at time $t' > t$ to carry out, say, a recovery action that may persist for a duration $\beta > 0$, where $(t' + \beta) < (t + \epsilon_i)$. In other words, the event handler should complete before, on the average, a next change in I_i occurs. The ϵ parameter depicts the time scale of event occurrences.

In the underlying execution of a distributed algorithm, the parameter $(\epsilon - \beta)$ may specify an upper bound on the time for collecting distributed snapshots of the system state.

4.2 Event predicates

In general, any type of critical property may be represented by a *condition* on the state variables $\Psi(S)$ exported by the external interface S that is visible to the application entities (i.e., users). An associated system event is said to occur when user-prescribed condition on the problem-specific state holds, i.e., a predicate $L(\Psi(S)) =$ true where $L(\cdots)$ is a logical formula. When the event occurs, it is understood that the system has indeed failed to meet the corresponding critical requirement. Some conditions may involve the state information at only one node, whereas some other conditions may involve many or all nodes.

An event e is said to occur at time t when a predicate *evnt_spec*(e) capturing certain type of changes in system interface state S becomes true in the interval $[t, t+\epsilon]$. These changes are observable by a distributed algorithm by sampling the interface state $\psi(S)$. Such event definitions may be given as:

$$evnt\_spec(e) \equiv L(\{(F_i, (\psi(S)_i = (A_i(t), k_i))\}_{i=1,\cdots,N}),$$

where F_i is an 'applicative' function that relates the observed value of an attribute A_i at time t with a constant k_i to yield a boolean result (such as $>$, $<$, $=$, $+$, $\cdots$). The functions $\{F_i\}$ may be hooked onto the computation as 'plug-ins' that can be invoked by the monitor to process the sampled sensor data as inputs[6].

The condition $L(\Psi(S))$ is supplied by a user to his/her local computation node. Once an event predicate is prescribed, a distributed 'event detection' algorithm is executed by monitoring nodes $\mathcal{M}$ to evaluate the predicate at various time points using the instances of interface state $\Psi(S)$ maintained by the computation nodes $\mathcal{C}$ (where $\mathcal{C} \bigcap \mathcal{M} \neq \emptyset$). The event is deemed to have occurred when the predicate evaluates to true, whereupon, this result should be propagated to appropriate nodes $\mathcal{C}$ (so that a recovery may be initiated).

The occurrence of events will be determined by applying the logical formula $L(\Psi(S))$ on a composition of the states maintained by computation nodes $\mathcal{C}$[7].

4.3 Illustrative examples of our specification approach

Consider the sensing of temperature p in a boiler plant. Suppose a critical situation is stipulated as the rate of increase of temperature exceeding a threshold T_2 when the current temperature is above T_1 (this may trigger the opening of a coolant liquid valve so that an explosion can be avoided). This event may be expressed as:

$$evnt\_spec(\text{EXPLODE}) \equiv (avg(p) > T_1 \wedge \frac{dp}{dt} > T_2).$$

Suppose the valve which was opened needs to be closed when the temperature falls below a threshold T_1' $(< T_1)$ — which implies that $\frac{dp}{dt} < 0$ — to avoid a

dural interfaces [4]. Monitors are then objects that detect the occurrences of events in the problem-domain.

[6]Our functional approach to specifying interface behaviors achieves more modularity than the 'state transition' based approach employed in [6] for database-like heavy objects.

[7]Sensor data from external environment may be made available to $\{F_i\}$ through auxiliary measurement tools which, in a programming sense, are outside the monitor proper.

freezing[8]. This relation may be expressed as:

$$evnt\_spec(\text{FREEZE}) \equiv (\{(avg(p) < T_1'\} \wedge \{(\frac{dp}{dt} < 0)\})).$$

The physical device that samples p is embedded into the snapshot-taking mechanism which filters the timed samples from the device.

Consider the earlier example of determining the 'inference reliability' parameter for a bank of sensors that detect enemy movements on the battle terrain. Treating the bank of sensors as providing a 'surveillance service', a critical property of this service may be prescribed as:

$$evnt\_spec(\text{SENREL}) \equiv (N_s > 10 \wedge \frac{N_g}{N_s} > 0.7),$$

where N_s is the number of sensors in the bank and N_g is the number of non-faulty sensors. N_s and N_g constitute the programming-level state visible at the service interface, in terms of which the event depicting a drop in the inference reliability to unacceptable levels is prescribed. The underlying protocol mechanisms to determine N_g from a knowledge of the set of sensors in the bank (such as 'majority voting' among sensors) is insulated from this interface specification[9].

Once the above form of predicates is loaded into the event monitor algorithm, the latter mechanically takes a snapshot of the system state maintained at various nodes — $(p, \frac{dp}{dt})$ and (N_s, N_g) respectively. It then computes the applicative functions — '$avg(\cdots)$'/'$\frac{d(\cdots)}{dt}$' and '$>$'/'$\frac{(\cdots)}{(\cdots)}$' on p and N_s/N_g respectively — to evaluate *evnt_spec*(EXPLODE) and *evnt_spec*(SENREL), as the case may be.

4.4 Event causality based system behaviors

Every event monitored at the system interface level is associated with a logical time of occurrence, namely, processing the change in interface state caused by an event e in the context of that caused by an event e'. We denote this relation between e and e' as $e' \succ e$. Due to involvement of human elements in target computation, the events can be observed only in an order given by the $\succ$ relations among them (i.e., causal order) [7, 8]. For instance, intrusion at a security classification level $j+1$ can occur only after that at a level j, where $j = 0, 1, 2, \cdots$ with increasing values of j indicating higher classification levels. This may be denoted as: intrusion(j) $\succ$ intrusion($j+1$). The causality relations between events are specific to the given problem domain (but may be expressed using a common notation).

Consider the earlier example of boiler plant temperature control. The *open_valve* and *close_valve* operations may be supported by a recovery module as part of the temperature control. Even though the enabling conditions for *open_valve* and *close_valve* are prescribed separately (by EXPLODE and FREEZE events respectively), there is an application-specific causal relationship that needs to be enforced on these actions, namely, *open_valve* $\succ$ *close_valve*. If a node sees these actions in the wrong order, it thinks that the valve is opened, while nodes seeing the actions in the causal order will treat the valve as closed. This may in turn lead to hazardous situations in the plant.

Thus the behavior B of a system interface may be prescribed in terms of the allowed temporal ordering relationships between various events possible at the interface level, given as: $B \equiv (\succ, \{e\})$.

4.5 Global snapshots for event recognition

The event monitor executes a distributed algorithm to evaluate *event_spec*(e) and determine if the event e has occurred. The event monitor should be non-intrusive on the target computation in that taking a snapshot should not interfere with the flow of computation. Such an algorithm falls under the class of 'global snapshot taking' that has been studied by researchers in a variety of contexts [9]. Currently available algorithms however base their decision upon taking a 'consistent cut' along the process timeline using Lamport's logical clocks. In our approach however, snapshot taking is elevated to an object-oriented message-passing level that employs predicate evaluation as the basis for event detection.

Here, consistency levels in event detection that suit the specific problem-domain can be realized. For example, congestion detection in a wide-area network setting does not require that the packet loss reports from receivers be exact. This is because the penalty

[8]That $T_1' < T_1$ implies a 'hysteresis' during decreasing trends of temperature, to avoid valve 'chattering'.

[9]For instance, an underlying protocol mechanism may employ a periodic polling of sensors (based on authenticated sensor identification) to determine N_s and a profile comparison of sensor outputs in response to simulated test inputs to to the sensors to determine N_g. A call from the application (say, through a GUI) to monitor sensor reliability will activate the protocol mechanism to collect N_s and N_g, and then a synchronization algorithm that timestamps the collected data and orders them chronologically to determine if the event SENREL has occurred.

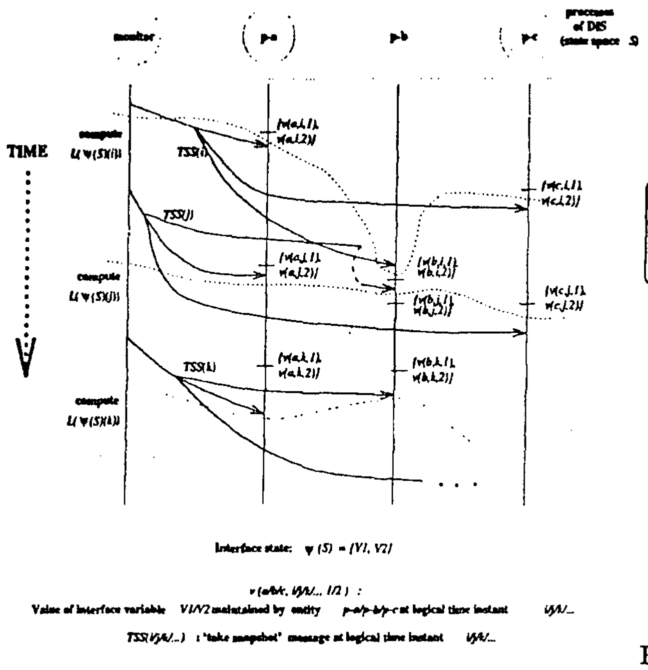


Figure 3: Illustration of snapshots of system state

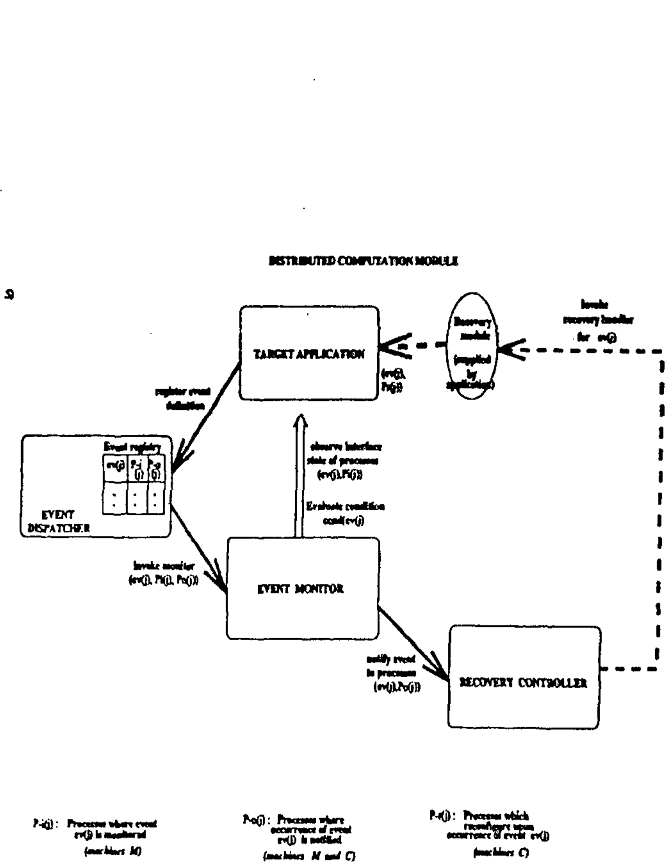


Figure 4: Computational view of monitor-based structure of DIS

for a coarse and/or inaccurate recovery often may not be high (such as attempting to reduce the sending rates when the reported congestion has ceased). Such problem-specific consistency requirements will be incorporated into the event detection algorithm.

Synchronous broadcast of messages between nodes $\mathcal{C}$ enables a structuring of event observations in which the monitor module does not need to sift through a large number of intermediate states traversed by $\mathcal{C}$, which otherwise will be needed if an asynchronous approach is employed. This is because, synchronous broadcasts implicitly satisfy the causality of interactions between nodes $\mathcal{C}$. This in turn obviates the explicit need to track causality at message-level actions by the nodes $\mathcal{M}$ (such as 'send' and 'receive' of messages), which is otherwise needed with asynchronous approaches. This in turn allows reducing the number of 'snapshot taking' for event detection (because we look only at a subset of Lamport's 'cut lines' where the predicate can become true). See Figure 3.

We note here that works have been carried out elsewhere on global event detection mechanisms [10, 11, 12]. In comparison to these works, tne amount of state tracking built into our snapshot protocols is less. Secondly, our protocols are general-purpose and canonical in nature, in that they can detect *user-profilable* events, and hence our protocols are applicable across a variety of problem-domains.

Overall, we view our approach as *event filtering*, wherein an event filter is a programmable selection criterion for classifying or selecting events from a stream of events. The event filter is basically prescribed in a high-level declarative language that is compiled into the event monitor portion of the programming system.

5 Software realization of monitor

There are four modules in our model of reactive programming of monitor: the 'event transformer', 'event detector', 'event dispatcher', and 'recovery controller'. See Figure 4. Each of these modules is distributed, with the processes implementing the distributed instances co-residing with those implementing the target computation. We have realized this execution model of monitor, as described below.

5.1 Functional components of monitor

The 'event transformer' maps the physical manifestation of an event into a form that can be observed by the monitoring mechanism. The event dispatcher maintains a registry of uninstantiated predicates supplied by the application (the registry may be viewed as an 'event rule library'). It invokes the event monitor with a predicate definition and appropriate parameters. The monitor performs *non-intrusive* observations on the target computation to detect the occurrence of events (i.e., the activity of monitoring does not interfere with the computation).

When a predicate becomes true signifying the occurrence of an event, the monitor collects the state parameters from various processes as required for recovery. The monitor then invokes a *recovery controller* that determines the temporal ordering of system-level reconfigurations through timed actuator signals. The latter can then be fed into problem-specific recovery modules through 'call back' hooks on exception handlers supplied by the target application (compiled into the programming system).

In our present implementation, we focus only on the event transformer, detector and dispatcher modules. Recovery control can be incrementally added on top of our monitor module.

5.2 Monitor implementation on HP OpenView

We have currently implemented the monitor software on 3 SUN Ultra-10 workstations interconnected by Ethernet. The monitor and computation modules typically co-reside in each workstation, interacting with one another through 'public class' routines. These routines are implemented through a JAVA API, providing access to event registry management and event detection mechanisms. JAVA Remote Method Invocation is employed for a distributed realization of these modules [13]. The APIs exercise HP OpenView primitives to make use of the latter's built-in management and reporting capabilities [15], for implementing our model of event monitor.

Unlike the implementation of a synchronous programming paradigm by combining JAVA threads and messages in the context of object-oriented programming as described in [14], our use of JAVA threads and messages will be to implement snapshot protocols in a highly asynchronous setting in order to reap problem-specific concurrency.

5.3 Menu-driven GUI for event specification

We have developed an interactive menu-driven window interface (GUI) between human users and the machines implementing the monitor system. The GUI allows users to inject a variety of critical scenarios in

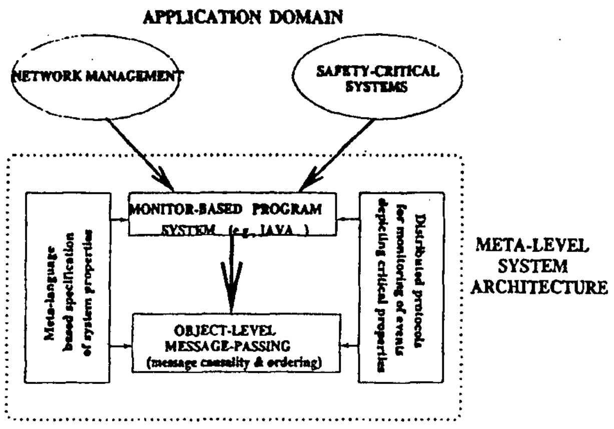

APPLICATION DOMAIN
NETWORK MANAGEMENT
SAFETY-CRITICAL SYSTEMS
MONITOR-BASED PROGRAM SYSTEM (e.g., JAVA)
OBJECT-LEVEL MESSAGE-PASSING (message causality & ordering)
Meta-language based specification of system properties
Distributed protocols for monitoring of events depicting critical properties
META-LEVEL SYSTEM ARCHITECTURE

Figure 5: Architectural layers in our programming system

the problem-domain being modelled (e.g., 'battlefield' simulations).

The menus basically identify the set of system tasks as a set of 'buttons' displaying the verbose description of the physical events and state variables. When a user clicks on a button, a 'value' menu space pops open, indicating the possible values for the task type clicked. A set of applicative functions prescribed by the computation are also be loaded into the desktop as icons. Using these buttons and icons, a user can profile an event as satisfying any desired condition. Users can profile different system properties by clicking on these 'buttons' and invoking pre-loaded applicative functions in the desktop to operate on the variables. The level of user participation is left to his/her ingenuity and extent of computer knowledge.

It is possible that some of the application programmers involved in the specification of events and event orderings may have less experience in distributed programming and algorithmic techniques. The user-friendliness of the menu-driven window interface to the event monitoring system may allow even such programmers to develop and study models of embedded systems.

Figure 5 gives an architectural view of our monitor-based programming system.

6 Conclusions

Monitoring of distributed real-time information networks to enable recovery from anomalous and faulty situations is the focus of our paper. We support this monitoring by employing an object-oriented specification approach to characterize system behaviors, and determine the monitorable events therefrom. We detect deviations from expected critical behaviors of a DIS as symptoms of information attacks (that may arise from component failures and/or external intrusion).

Our new idea is the prescription of critical properties a DIS should satisfy as a set of logical formulae. These formulae can be transcribed into event predicates on the computation state, which can then be monitored by a distributed snapshot algorithm. The programming model comprises of a notation for specifying events and a composition mechanism for relating various parts of system interface state into events. The model exercises a decentralized structure of protocol agents at target computation nodes that monitor events during a program execution.

From a distributed programming perspective, our monitor-based structure of a DIS underscores two elements:

- Integration of the 'flow of real-time' into program-level notion of 'sensors' and 'actuators';
- Meta-level specification of events in terms of generic operators and applicative functions.

The meta-level specification method can be employed in a variety of application domains.

The monitoring system we have developed allows the 'plug-in' of a variety of target computations to carry out their problem-specific tasks. The 'plug-in' manifests as a declarative specification of interface behaviors of these tasks and instantiating the event monitor algorithm with an appropriate set of parameters, i.e., the prescription of event predicates and causality relations between events. A user-level specification of interface behaviors is incorporated through a GUI.

Overall, our programming model underscores a modular decomposition approach to monitoring distributed information systems, and hence makes their verification easier. This in turn reduces the cost of developing and maintaining Complex Distributed Embedded Systems.

References

[1] S. Chamberlain. **Automated Information Distribution in Bandwidth-constrained Environments.** In Proc. *Milcom'94*, North-Holland pub., 1994.

[2] R. R. Brooks and S. Iyengar. Chap. on **Sensor Fusion and Approximate agreement.** In *Multisensor Data Fusion*, Prentice-Hall Publ., 1998.

[3] B. Dutertre and V. Stavridou. **Formal Requirements Analysis of an Avionics Control System.** In *IEEE Transactions on Software Engineering*, vol.23, no.5, pp.267-277, May 1997.

[4] P. Felber, R. Guerraoui and M. E. Fayad. **Putting OO Distributed Programming to Work.** In *Communications of the ACM*, pp.97-101, vol.42, no.11, Nov. 1999.

[5] H. Kopetz and P. Verissmo. **Real Time Dependability Concepts.** Chap. 16, *Distributed Systems*, ed. S. Mullender, Addison-Wesley Publ. Co., 1993.

[6] S. S. Lam and A. U. Shankar. **Specifying Modules to Satisfy Interfaces: a State Transition System approach.** In *Distributed Computing*, Springer-Verlag, vol.6, 1992.

[7] K. Birman and T. A. Joseph. **Virtual Synchrony in Distributed Systems.** In Proc. *Symp. on Operating Systems Principles*, ACM-SIGOPS, 1987.

[8] K. Ravindran and S. Samdarshi. **A Flexible Causal Broadcast Communication Interface for Distributed Applications.** In *Journal of Parallel and Distributed Computing Systems*, vol. 16, no. 2, pp. 134-157, Academic Press Publ. Co., Oct. 1992.

[9] K. M. Chandy and L. Lamport. **Distributed Snapshots: Determining Global States of Distributed Systems.** *ACM Transactions on Computer Systems*, 3(1):63–75, Feb. 1985.

[10] R. Cooper and K. Marzullo. **Consistent Detection of Global Predicates.** In Proc. *Workshop on Parallel and Distributed Debugging*, ACM, pp.167–174, 1991.

[11] V. K. Garg. **Observation of Global Properties in Distributed Systems.** In Proc. *Intl. Conf. on Software and Knowledge Engineering*, IEEE CS, pp.418-425, Lake Tahoe (NV), June 1996.

[12] E. Fabre, A. Benveniste, and C. Jard. **Distributed State Reconstruction for Discrete Event Systems.** In *Tech. Report* CDC00-REG1625, IRISA (France), March 2000.

[13] B. Krupczak, K. L. Calvert and M. H. Ammar. **Implementing Communication Protocols in JAVA.** In *IEEE Communications*, Oct. 1998.

[14] C. Colby, L. J. Jagadeesan, R. Jagadeesan, K. Laufer, and C. Puchol. **Objects and Concurrency in Triveni: A Telecommunication Case Study in Java.** In proc. COOTS, 1998.

[15] HP WBM. **HP Proactive Networking: The Networking Management Component.** Hewlett-Packard white paper, 1998.

To appear in proceedings of 1st International Conference on Communication System Software and Middleware (COMSWARE 2006), New Delhi (India), January 2006. (sponsored by IEEE Com. Soc. and ACM-SigMobile)

Appendix 5

'dynamic protocol plug-in': a Middleware Provision for Enhancing Network Service Performance **

K. Ravindran & Jun Wu *

Abstract

The paper describes a protocol-level adaptation mechanism to enhance the performance of network service provisioning to clientele. In our model, a service provider (SP) maintains multiple protocol modules to exercise the infrastructure resources. Here, each protocol exhibits a certain degree of cost optimality (in terms of resource usage) in distinct operating regions of the network infrastructure. An example is the cost tradeoff between the 'goback-N' and 'selective repeat' based retransmission schemes for 'reliable data transfer' between end-points under various packet loss rates in the network. During run-time, the SP selects one of the protocol modules that can meet the client-requested service obligation under the prevailing resource-level operating conditions. Our model allows a 'dynamic switching' from one protocol module to another at run-time based on the changing network conditions. The overall goal is to offer a sustained access to the service with a resource-optimal and QoS-compliant service offering. The paper describes 'protocol switching' as an architectural foundation for building cost-effective network services.

* K. Ravindran and Jun Wu are with the Dept. of Computer Science, City University of New York (City College and Graduate Center), USA — Contact e-mail: *ravi@cs.ccny.cuny.edu; juwu@gc.cuny.edu.*

** Part of the work on dynamic service compositions and environment monitoring was supported by AFRL-HE grant: F33615-03-1-6385 (2003).

1 Introduction

A network service model allows client applications to choose the level of service required from a network infrastructure. The model allows for two types of entities: a *service provider* (SP) and a *service user* (or client). The SP implements a protocol mechanism, running on different sites of a communication network, to provide the client-requested service. The actual delivery of service to clients involves the use of infrastructure resources at these sites in a way determined by the protocol mechanism. The current management standards, TINA and DCOM [1, 2], advocate this model to integrate network service management tools into application development environments.

In the current models, a service offering is exported to the clientele through an abstract interface that hides the underlying protocol-level details. Client applications may prescribe service requirements in terms of QoS parameters exported through the interface. A protocol module may then be instantiated by the SP with the client-prescribed QoS parameters. Though the models are good from a software engineering standpoint, their performance potential are often constrained by the difficulty in adapting the protocol operations as the service operating points change dynamically. We elaborate on this below.

The interactions between the protocol operations and the infrastructure resource usage exercised therein to provide a client-requested service are often complex. For example, a sliding window protocol provides a certain 'end-to-end data transfer rate' by a suitable choice of window sizes and packet retransmission policies under a given infrastructure-level network bandwidth and packet-loss behavior. A closed-form relationship between the various 'data transfer' parameters (say, determining the window size to achieve a certain transfer rate) is however possible only under simplifying assumptions about the protocol and the network. The other side of this complexity is the fact that different protocols are cost-effective (i.e., incur low overheads and resource consumptions) in different operating regions of the network. In the 'data transfer' example, a'goback-N' retransmission policy is efficient when packet loss in the network is small; whereas, a 'selective repeat' policy is efficient when packet loss is high. Thus, from a cost-effectiveness standpoint, the current models of SP-level control of protocol operations and infrastructure resources are often ad-hoc.

In this paper, we describe a new model service pro-

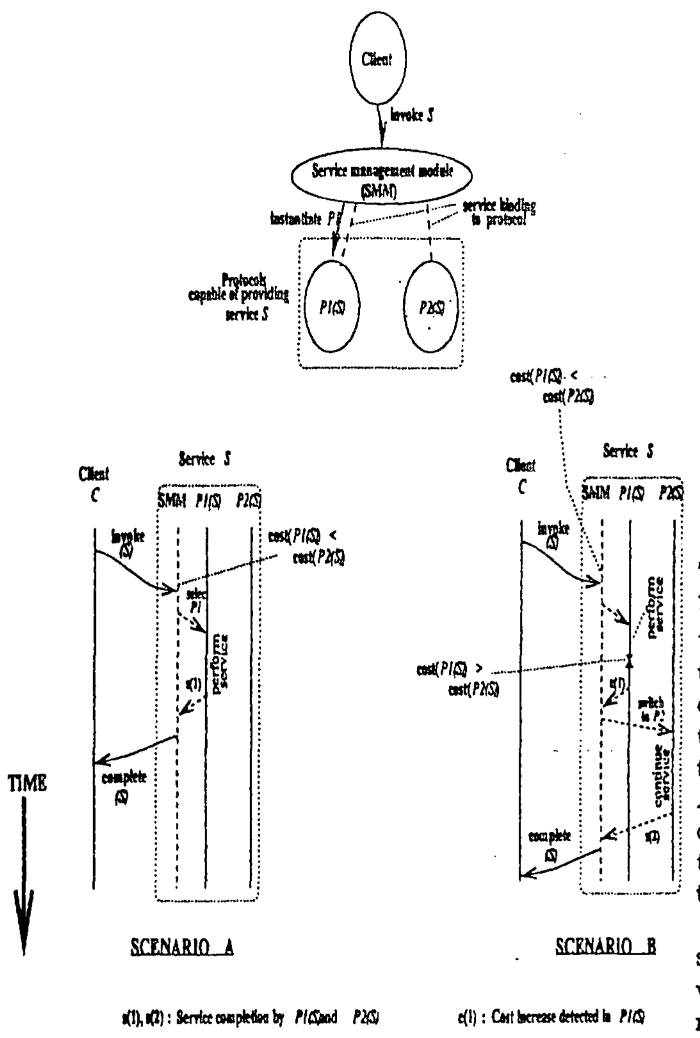

Client
invoke S
Service management module (SMM)
instantiate P1
service binding to protocol
Protocols capable of providing service S
P1(S)
P2(S)
cost(P1(S)) < cost(P2(S))
Client C
Service S
SMM P1(S) P2(S)
invoke (S)
select P1
perform service
s(1)
complete (S)
cost(P1(S)) > cost(P2(S))
switch to P2
continue service
s(2)
TIME
SCENARIO A
SCENARIO B
s(1), s(2) : Service completion by P1(S) and P2(S)
c(1) : Cost increase detected in P1(S)

Figure 1: Our model of network service offering

visioning using principles of functional decomposition of service components across well-defined interfaces. Our goal is to improve the performance of service offerings to clientele, i.e., offer a network service with low overheads and resource consumptions on the infrastructure while meeting the client-prescribed QoS. The SP may maintain multiple protocol modules, with each module optimized for performance over a distinct operating region of the service and the infrastructure. The SP may dynamically select one of the protocols and instantiate it with the given service parameters.

A *service-level management module* (SMM) maintains the binding information for various protocol modules provided by the SP. In a dynamic setting where depletions in infrastructure resources and changes in external operating conditions occur at various points in time, the SMM-level binding information may reflect these changes onto the appropriate choices of protocol modules to enable a resource-optimal service provisioning to clients. Figure 1 shows a high-level view of the role of SMM using two protocols P_1 and P_2. The SMM monitors the current service-level operating point, and determines the resource costs incurred by P_1 and P_2. The SMM then chooses P_1 or P_2, whichever incurs a lower cost, to provide the service.

To achieve cost-effectiveness of service offerings, two key functional elements are embodied in the SMM:

1. A *protocol switching* feature that allows a form of 'hot-swapping' of protocols at run-time;

2. An *environment monitoring* mechanism that can determine the operating conditions currently prevailing in the network.

The SMM is thus an active part of a SP, liaisoning between client applications and the underlying network protocols to offer the required levels of service under various environment conditions. Our paper focuses on a *meta-level description* of the SMM functions that embody management-oriented interfaces to the domain-specific functions implemented by the SP. A procedural realization of the SMM is itself based on our studies of different types of network applications: 'content distribution networks' (CDN) and end-to-end QoS-controlled transport.

The paper is organized as follows. Section 2 describes a service-oriented model of distributed networks based on protocol compositions. Section 3 augments the model with a support for dynamic protocol plug-ins to enhance system-level performance (i.e., lower resource costs). Section 4 describes a sample case study: adaptive CDN management. Section 5 relates our approach with the existing network management models. Section 6 concludes the paper.

2 Our model of network services

The functions of a network sub-system are made available to client applications through an abstract *service interface* that prescribes a set of well-defined capabilities (or service features). A client may exercise one or more of these features to obtain a certain quality of service delivery. For example, bounding the page access latency in a content distribution network (CDN) is a feature of 'content access' service that can be invoked by a web client by specifying an acceptable latency L as parameter. The CDN service interface maps the L parameter onto the underlying 'page access' protocol mechanisms, such as the page replica placement in the distribution network [3]. The choice of actual protocol mechanisms to provide a given level of service is itself hidden from the application.

2.1 Function-oriented view of services

A protocol module $P(S)$ embodies a policy function F that maps the application-specified QoS parameters a onto a resource allocation r in the network infrastructure under the current operating conditions e of an external environment. Here, e depicts the conditions and activities that occur outside the realm of protocols and application-level users but which impact the QoS delivered to applications. The QoS-to-resource mapping may be denoted as:

$$r = F(a, e) \quad \text{for } a \in \mathcal{A}, e \in \mathcal{E};$$

where $\mathcal{A}$ depicts the QOS parameter space visible to applications and $\mathcal{E}$ depicts the meaningful operating regions of external environment. Alternately, given a resource allocation r and environment condition e, a depicts the estimate of QoS deliverable to applications. The changes in a and/or e occurring over time constitute 'events' that drive the SP's estimation of resource needs r. The SP may use a suitable policy function and a protocol-level allocation of r in the infrastructure over meaningful time-scales. A procedural realization of resource allocation r is carried out by the domain-specific distributed state-machine elements of $P(S)$ executing at various control nodes of the infrastructure network.

The resource allocation function F exhibits a monotonic behavior with respect to the user-level QoS specs and environment conditions, denoted as:

$$F(a + \delta a, e) > F(a, e) \;\text{ and }\; F(a, e + \delta e) > F(a, e);$$

where the '+' operator is defined over the QoS parameter space and the external environment conditions, as the case may be. A change in QoS specs, say, from a to $a - \delta a$, is modeled as a *user-induced event*. It depicts client adaptations in response to either the network notifications of resource depletion or the application-induced reductions in a. Here, the protocol $P(S)$ needs to estimate the reduced resource needs $F(a - \delta a, e)$ under the current operating conditions e. On the other hand, a change in the environment conditions, say, from e to $e + \delta e$, is modeled as an *external event*. It depict an increase in the hostility of environment conditions that needs to be countered by $P(S)$ with an increased resource allocation: $F(a, e + \delta e)$. This event-driven functional view of network services is shown in Figure 2.

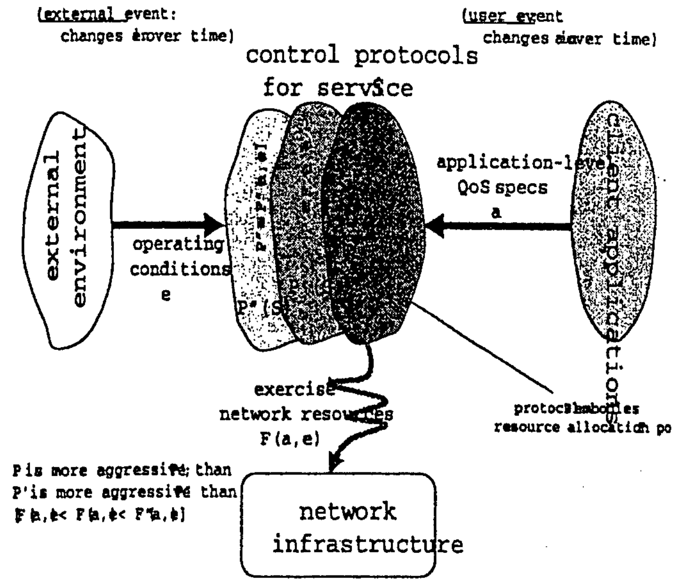


Figure 2: Event-based functional view of services

In a CDN for example, the service behavior depicts how the observed latency in page access L' varies relative to the prescribed L parameter under various client access and server content change patterns and distribution network characteristics. Here, the rate of client requests (f_c) and rate of server content changes (f_s) and the link bandwidth/delays and physical topology of network (such as node degree) are part of the external environment conditions $\mathcal{E}$ that impacts L'. $P(S)$ depicts the content access protocol employed by the SP that determines the placement of page replicas and when/how to update them, given a geographic location of clients relative to the content server. And, r depicts the replica placement configuration in terms of the network topology parameters.

From a management standpoint, the SP makes a network service *adaptive* by dynamic instantiations of a protocol $P(S)$ with the a and e parameters to drive the native functional elements of $P(S)$ for exercising the underlying network infrastructure resources.

2.2 Protocols as policy functions

F may be drawn from a repertoire of policy functions $\mathcal{F}$ that offers a range of resource allocation profiles for a given QoS specs and external conditions, i.e., $F \in \mathcal{F}$. The procedural realization of various policies: $F, F', F'', \cdots$ may itself be embodied into distinct protocols $P(S), P'(S), P''(S), \cdots$ implemented by the SP.

F may possibly be a coarse representation of the exact mapping, i.e., the actual QoS delivered may be a' instead of a, where the goal of SP is to ensure that $a' \approx a$. In some complex distributed networking applications, a closed-form representation of F may not even be feasible (in a computational sense) due to the large problem dimensionality of QoS-to-resource mapping. In the example of CDN, the page access latency may depend on a variety of factors: such as the geographic placement of proxy caching nodes relative to clients and servers in the network topology, the processing and storage capacity of proxy nodes, the link bandwidths/delays between nodes, and the client-generated access traffic patterns. From a service-level management standpoint however, the complex interactions between the QoS delivered to users and the infrastructure resource needs should be captured through at least a coarsely representable form of F.

Where a closed-form representation of F does not exist, the SP may employ a combination of short-range extrapolation of resource allocation estimates (e.g., linear predictors) and an agent-based monitoring of a' over short time-scales in response to these estimated allocations. The monitoring enables a 'control-theoretic' adjustment of resource estimates by the SP for converging onto a final accurate resource allocation — which makes $a' \approx a$. The faster time-scales in the agent-based monitoring and adjustment of a' are however unnoticeable to the client applications, since the latter operate at much slower time-scales.

Regardless of how a QoS-to-resource mapping is determined in an application domain, the mapping is representable as a functional relation $r = F(a, e)$ for use by the SP in revenue-driven resource management decisions. The resource allocation r is itself expressed in terms of protocol internal parameters.

We now illustrate our functional view of network services with a simple example of 'reliable data transfer' between end-points[1].

2.3 Example: 'reliable data transfer'

Consider a 'reliable data transfer' service (DAT) between two end-points c_1 and c_2 over a network. In a management view, c_1 and c_2 are the application entities prescribing a required end-to-end transfer rate U. The bandwidth apportionment B in the underlying network path, say, from c_1 and c_2 that allows the scheduling of packet flow over a shared network infrastructure constitutes the resource allocation. The packet loss phenomenon occurring along this path — measured as a loss rate l — impacts the achievable end-to-end transfer rate, thereby constituting the external environment for 'data transfer' protocols. $P(\text{DAT})$ may be a window-based protocol that transmits a set of packets w from c_1 to c_2 before awaiting acks and resorts to a time-out based retransmission of one or more packets of w to recover from potential packet loss. F may depict, say, a 'goback-N' policy for packet retransmissions, or a 'selective repeat' policy. $P(\text{DAT})$ is basically a procedural realization of F with distributed state-machine functions placed at c_1 and c_2: such as packet sequence numbering, buffering of packets of w at c_1 and/or c_2, generation of acks for packets received at c_2, and setting up of timers.

An estimate of the required bandwidth allocation along 'data transfer' path: $B = F(U, l)$, is given by:

$$B = \frac{U\overline{v}}{w},$$

where $\overline{v}$ is the mean number of transmissions required to deliver w packets to the receiver end-point ($\overline{v} > w$ for $l > 0$). The transfer efficiency $\frac{w}{\overline{v}}$ depends on the retransmission policy and the external environment parameter l (i.e., packet loss rate) — besides other protocol-internal parameters such as the timeout period for packet retransmissions and the packet sizes [4]. With a simplifying assumption that the ack packets are small enough to be immune from loss, the 'goback-N' scheme has $\overline{v} = \frac{1+\eta l}{(1-l)}$, where $\eta > 0$ is a normalized network delay parameter; and the 'selective repeat' scheme has $\overline{v} = \frac{1}{(1-l)}$. Such a computation of $\overline{v}$ is embodied in the applicative functions maintained by the SP that represent the retransmission scheme employed in the 'data transfer' protocol[2].

For a given w, the bandwidth allocation incurred by the 'selective repeat' policy (B_{sr}) to achieve a prescribed U is less than that by the 'goback-N' policy (B_{gb}), but requires an additional buffering of w at c_2 to re-order the received packets in the presence of packet misses. Accordingly, resource allocations in the 'selective repeat' and 'goback-N' policies may be given as $[B_{sr}, w]$ and $[B_{gb}, 0]$ respectively (with suitable normalization constants).

For management purposes, the changes in client specs of U and the changes in packet loss rate l are

[1]'reliable data transfer' protocol is well-known among network practitioners. We use this protocol merely as an illustrative example to highlight the subtle elements of our management-oriented network service model.

[2]Our focus is not on the accurate formula to estimate $\overline{v}$, but is on expressing the formula for $\overline{v}$ in a closed-form, to enable its on-line computation using an applicative function in the SMM.

modeled as events that drive the DAT SP's operations. The SP may obtain the external parameter l either by a monitor integrated into the 'data transfer' protocol or by an operator-controlled input through a management GUI.

We now describe how our function-based model of network services can be adopted by a SP to realize cost-effective service offerings.

3 Management operations of SP

We consider only the external events depicting changes in environment conditions, say, from e to $e + \delta e$, and the attendant operations of SP to optimize its revenue-driven resource allocations (while meeting a constant user-level QoS specs a). It involves a continuous monitoring of the external environment conditions and a dynamic selection of protocol mechanisms to deal with the prevailing conditions in a cost-effective manner. The management aspects pertinent to these functions are described below.

3.1 Client-visible service behaviors

How the service 'quality' as perceived by a client varies with respect to changes in the infrastructure resources constitutes a service behavior. Given a QoS expectation $a \in \mathcal{A}$ prescribed by the client on a service S, a protocol module $P(S)$ exercises the infrastructure resources r in providing a level of service $a' \approx a$ under a given external environment condition $e \in \mathcal{E}$. The resource allocation is represented as a function: $r = F(a, e)$, where $F(\cdots)$ abstracts the allocation policy embodied in $P(S)$. Here, a and r depict quantifiable indices: a being user-oriented and r being network-oriented, with the dynamic changes in e (i.e., external events) driving the SP's operations. The granularity of event detection mechanisms and their time-scales are controlled by the SP based on revenue-driven incentives.

In a CDN for example, the service behavior depicts how the observed latency in page access L' varies relative to the prescribed L parameter under various client access and server content change patterns and distribution network characteristics. Here, variations in the rate of client requests (f_c) and rate of server content changes (f_s) and fluctuations in the link bandwidth/delays and physical topology of network (such as node degree) are part of the external event space $\mathcal{E}$ that impacts L'. $P(S)$ depicts the content access protocol employed by the SP that determines the placement of page replicas and when/how to update them, given a geographic location of clients relative to the content server. And, r depicts the replica placement configuration, expressed in terms of the network topology parameters.

For a given amount of infrastructure resources r, the various protocol modules $P(S), P'(S), \cdots$ capable of providing S may exhibit distinct behavioral profiles, i.e., offer different levels of service S. This arises due to the distinct ways in which each protocol uses the infrastructure resources. In the CDN example, the case of $f_c \ll f_s$ causes a pull-based protocol (i.e., client-driven update) to incur higher access latency (but lower replica update overheads), in comparison to a push-based protocol (i.e., server-driven update). The increased latency arises because of the need to check if a replica is up-to-date relative to the server before a pull, even in the absence of server-initiated updates of replicas.

From a client's perspective, two services are identical if their visible behaviors for a given a set of control parameters are the same. In general, service differentiation may be in terms of the measurable parameters of client-visible behaviors. When multiple protocol modules $P(S), P'(S), \cdots$ are capable of providing a service S, the choice of a candidate protocol, say, $P(S)$, should be based only on the run-time resource demands exercised by $P(S)$.

3.2 Monotonicity properties of F

To enable dynamic service provisioning, the SP needs a quantitative handle on how the functional relation $r = F(a, e)$ that depicts the domain-specific QoS-to-resource mapping varies with respect to e. $F(a, e)$ may be modeled as a strictly monotonic *convex* function with respect to e, denoted as:

$$\forall a \in \mathcal{A}\ [F(a, e_2 + \delta e) - F(a, e_2)] > [F(a, e_1 + \delta e) - F(a, e_1)] \text{ for } e_2 > e_1.$$

The convexity of $F(a, e)$ depicts that a given increase in the hostility of environment conditions can only be countered with incrementally higher resource allocations. It captures the 'multiplier effects' caused by an additional level of hostility in the environment that requires diverting some of the erstwhile-allocated resources. In the example of 'reliable data transfer', an increase in packet loss rate l incurs a higher-than-linear increase in bandwidth allocation B to account for the additional packet retransmissions caused by the loss of a packet in successive transmission cycles. Alternately, the convexity depicts that a protocol $P(S)$

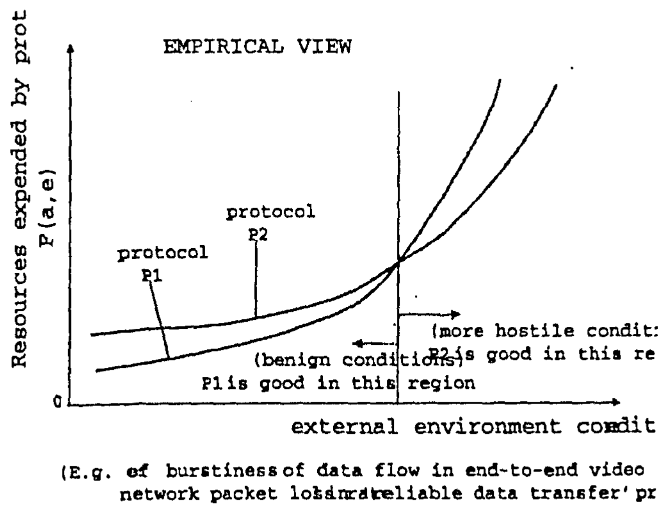


Figure 3: Convexity of resource allocations

embodying the policy $F(a,e)$ is more optimal in resource usage at lower values of e, with the optimality decaying rapidly at higher values of e.

The protocols $P(S), P'(S), P''(S), \cdots$ may exhibit different levels of monotonic behavior in different operating regions of the $\mathcal{E}$-space — i.e., the rate of change in the optimal behavior with respect to e is different for $P(S), P'(S), P''(S), \cdots$. So, a protocol $P(S)$ that is resource-optimal in one region may not be so in another region where $P'(S)$ may be optimal, and so on. In the 'reliable data transfer' example, the 'goback-N' policy is efficient when packet loss rates are low, whereas the 'selective repeat' policy is efficient at higher loss rates. Thus, the convexity of protocol costs, combined with the short-range optimal behaviors, can make a currently active protocol behave sub-optimally as the environment conditions e change. See Figure 3 for an illustration.

As seen, the monotonic convexity of $F(a,e)$ can be concretely captured through the domain-specific parameters representing e. Since the external events occur spontaneously (i.e., cannot be predicted or controlled by the SP), it is necessary to monitor the environment conditions e continuously to enable the SP's dynamic switch to a resource-optimal protocol.

3.3 Protocol switch vs parameter change

A protocol $P(S)(par)$ encapsulates a infrastructure resource scheduling functionality, with the parameters par instantiating this function at run-time to control the extent of resource scheduling. In other words, $P(S)$ prescribes a policy on how to schedule the resources, whereas par controls the extent of resource scheduling under the given policy. We exemplify this below with observations drawn from our earlier study of end-to-end video transport applications [5].

Consider two different protocol mechanisms to realize QOS-controlled 'data connectivity' service (CONN): 'window-based flow control' (WIN) and 'rate controlled flows' (RATE) [6]. The use of window-based credits to send data packets over a path constitutes the WIN protocol. Here, the size of window w is the parameter for an instantiation: WIN(CONN)($\{w\}$). On the other hand, controlling the rate of data flow at various points in time constitutes the RATE protocol. Here, multiplicative constants α and γ to determine a rate increase and decrease respectively are the parameters for an instantiation: RATE(CONN)($\{\alpha, \gamma\}$). The WIN protocol is cost-effective, in terms of path utilization, when the data flow is less bursty — as with video frames containing low-motion scenes. Whereas, the RATE protocol is cost-effective when the data flow is highly bursty — as with high-motion scenes. Figure 4 illustrates this notion of protocol switching and parameter changing with a sample data flow scenario. Here, the level of burstiness in data flow depicts the external environment parameter e for the CONN protocols.

A protocol switch from $P(S)$ to $P'(S)$ manifests as a change in the resource scheduling mechanism as embodied in P and P' respectively. Whereas, a parameter change, say, from $P(S)(par_1)$ to $P(S)(par_2)$ merely adjusts the level of scheduling employed by P to effect a change from par_1 and par_2. When selecting a protocol $P(S)(par)$ to provide the service S, the parameters par, which reflect the infrastructure resources exercised by $P(S)$, are determined by the SP to meet the client-prescribed requirements a.

In summary, the representation of $F(a,e)$ in our service model is based on the notions of 'application QoS' and 'network resource', and the generalized 'economic theory' principles that govern the QoS-and-resource interactions [7]. A concrete representation is then incorporated by the SP in its service-level management procedures to optimize revenue accruals.

4 Case study of CDN service

In a 'content distribution network' (CDN) such as AKAMAI, a server hosts the content pages of an information base — e.g., electronic shopping catalogue. With a large number of clients accessing various pages (say, for downloading), the server maintains copies of

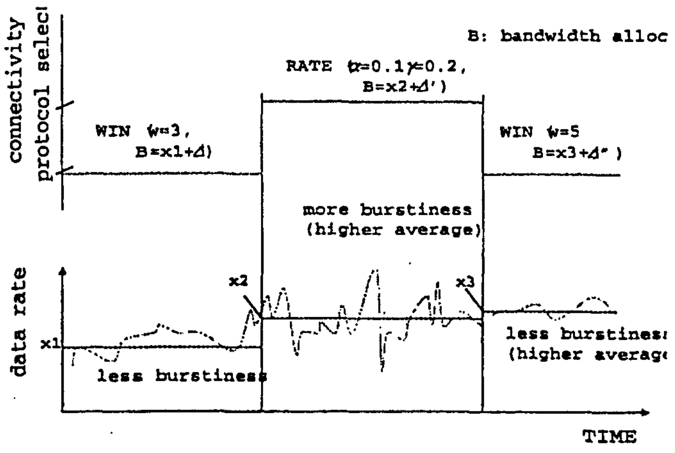


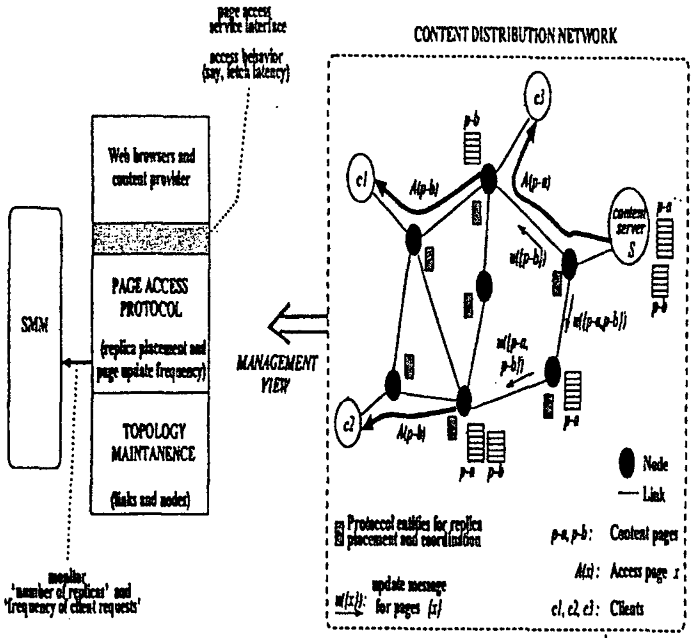
Figure 4: Illustration of protocol-level adaptation for QoS-controlled 'data connectivity'

the pages at multiple proxy nodes (or, replicas) in the distribution network through an update protocol. A client request is forwarded to the nearest proxy node containing the requested page. Keeping page copies of the information base at multiple replica nodes in the network increases access performance, i.e., reduces the access latency L' seen by clients. See [8] for a comprehensive discussion of adaptive 'content distribution' protocols from a service-level standpoint.

4.1 Management view of page updates

The functional elements in a CDN protocol are the server R where master copies of various content pages are maintained, the proxy nodes of R where mirror images of these content pages are placed, and the communication links in the path between a client C and the proxy nodes. There are two key components of the protocol: i) the placement of replicas in the network, and ii) the update policies to keep the replicas consistent. See Figure 5 for an illustration. The infrastructure resources are the network link bandwidths and the placement of proxy nodes in the network topology. The frequency of client access requests f_c and the frequency of server content changes f_s constitute the external environment parameter. Resource costs directly relate to the amount of proxy update message traffic and/or data movement through the network to deliver a content page to C. Clients prescribe a tolerable access latency L as the expected QoS, with the actual latency $L' < L$.

Given a page update policy, the geographic spread-out of replicas relative to the location of C in the network determines the latency L' of page fetches by C



page access service interface
access behavior (say, fetch latency)
CONTENT DISTRIBUTION NETWORK
Web browsers and content provider
PAGE ACCESS PROTOCOL
(replica placement and page update frequency)
TOPOLOGY MAINTANENCE
(links and nodes)
SMM
MANAGEMENT VIEW
monitor 'number of replicas' and 'frequency of client requests'
c1
c2
c3
A(p-b)
A(p-a)
content server S
p-a
p-b
u([p-b])
u([p-a, p-b])
Node
Link
Protocol entities for replica placement and coordination
p-a, p-b: Content pages
A(x): Access page x
u([x]): update message for pages [x]
c1, c2, c3: Clients

Figure 5: Management view of a CDN

and the message traffic through the network to to keep the page copies consistent. The replica placement and update mechanisms are embodied into the page access protocol that provides the CDN service. Typical policies for page updates include: i) a pull protocol, i.e., correcting the copy at a node when a client attempts to access this copy (client-driven update scheme CL), and ii) a push protocol, i.e., correcting the copies at various nodes whenever the server changes its master copy (server-driven update scheme SR)[3]. When client access patterns $\frac{f_c}{f_s}$ change (say, at different times of a day), the SP may switch the page update mechanisms accordingly to maintain cost minimality.

4.2 Protocol-level reflection of $[f_c, f_s]$

The infrastructure resource r depicts the placement of replica nodes (i.e., for content caching) in the network topology. The page access protocol may maintain a macroscopic representation of r with an internal parameter: 'degree of page replication' achieved using the replica nodes — denoted as *drep*. The environment parameter (f_c, f_s) determines the level of synchronization needed among replicas in the protocol — denoted as $sfreq$. For the CL protocol, $sfreq = f_c$, i.e., the frequency of client requests for page access.

[3]Another policy may be to associate a 'life-time' for each page, upon the expiry of which a node hosting a copy of this page obtains a fresh copy from the content server.

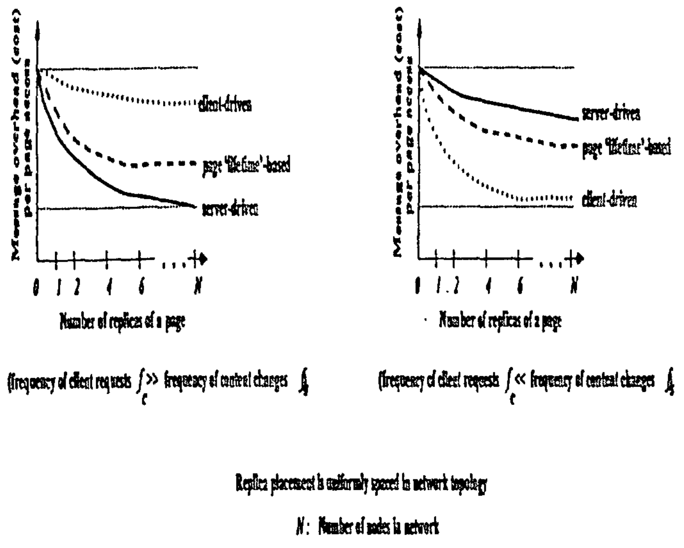


Figure 6: Cost comparison of update protocols

For the SR protocol, $sfreq = f_s$, i.e., the frequency of updates occurring on server-maintained pages. The[4] average message overhead (normalized) in accessing a page may empirically be given as:

$$msg = \frac{sfreq.drep^k}{f_c} \quad \text{for } 0 < k \leq 1.0,$$

with k depicting the actual replica placement in physical topology. The page access cost attributed to a client request may be determined therefrom.

The parameter *drep* is so chosen to keep the page fetch latency within the limit L prescribed by clients. For various values of *drep*, the page access costs may be compared for the CL and SR protocols, and then a cost-optimal protocol may be chosen by the SMM.

Figure 6 illustrates the normalized service-level cost variations with respect to the underlying protocol parameters for a CDN service. The cost illustration is based on empirical data collected from our studies of protocol mechanisms for the CDN service offering.

4.3 Monitoring of CDN parameters

Monitoring involves detecting the changes in: i) external environment parameters $[f_c, f_s]$, and ii) network-level topology parameters such as link bandwidth/delays and node degree. The monitoring of (i) may occur at a slower time-scale meaningful to the applications: say, once every 3 minutes. The monitoring of (ii) may occur at a faster time-scale: say, sampling the network parameters every 10 seconds. These time-scales are reflected onto the time granularity of SMM decision-making for CDN performance management.

When a mapping relation between latency and *drep*, i.e., $drep = F(L, [f_c, f_s])$, does not exist in a closed-form — such as Equation (4.2), the SP may resort to heuristics-based learning techniques to determine the (sub-)optimal *drep* — such as piece-wise linear estimators [3]. Furthermore, it is possible that a latency constraint is not satisfiable for a current set of network parameters. In these lights, the monitoring of L' allows determining if the current replica placement configuration in the network meets the client-prescribed latency constraint L.

To monitor L', client requests are time-stamped at the time of generation. The time elapsed for fetching a requested page at a client site X is the access latency experienced by X. A time-series of latency samples is collected at X over an observation interval, from which a smoothed latency value is obtained by an averaging algorithm. The latency value so estimated at X is then reported to the SMM. The latter collects the latency values of various clients $\{X\}$, and employs an extrapolation algorithm to obtain a time-synchronized network-wide view of L'.

Similarly, the monitoring of f_c involves sampling the number of page requests emanating from a client X over an observation interval. Therefrom, a statistical measure of the request rate of X (such as mean) is obtained. The SMM then combines this rate information collected from various clients $\{X\}$ to determine a time-synchronized view of f_c. The sampling of f_s is straight-forward, namely, the content server notifying the SMM at every update to the master content.

For simplicity, the paper assumes that the network topological parameters do not change (say, the nodes in physical topology do not fail and the bandwidth/delay along a link does not change). The assumption allows us to focus on the functional elements of protocol-level CDN adaptation, without compromising the generality of our model to accommodate network-level changes.

We now describe the meta-activities to switch between the CL and SR protocols at run-time (say, the user-level latency prescription L remains fixed).

4.4 Controlling replica updates

The CL scheme requires that each copy of a page carry the time-stamp of its last update (LTS) and the most recent update on this page that has occurred

[4]With 'life-time' based page replenishment, $sfreq$ may be determined by the life-time associated with pages.

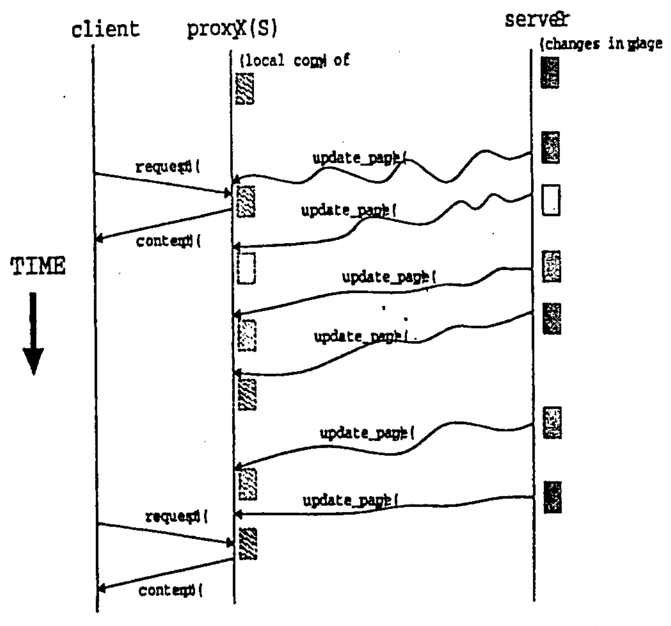


Figure 7: Server-driven update scheme (SR)

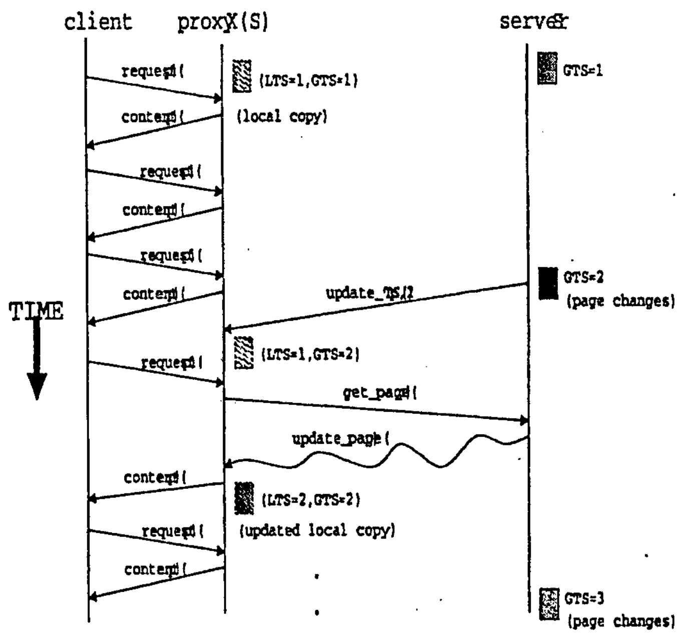


Figure 8: Client-driven update scheme (CL)

across the entire network (GTS). Suppose a client request arrives at a proxy node X for a page p. If LTS(X,p) = GTS(X,p), X sends its copy of p to the client. If LTS(X,p) < GTS(X,p) — meaning that X's copy of p is obsolete, X contacts an upstream proxy node Y that has LTS(Y,p) = GTS(Y,p), and then obtains the copy of p from Y to forward to the client. When the content server Z updates its master copy of p, it sets GTS(Z,p) to the current time, and then propagates this time-stamp to all the proxy nodes for p to update their LTS values. In the SR scheme, every proxy node has the most recent copy of p relative to that at Z. This is achieved by having Z propagate an update to all the replicas whenever there is a change in the master copy of p at Z.

Figures 7 and 8 illustrate the SR and CL schemes respectively.

4.5 A scenario of CL-SR switching

The parameters that influence the choice of CL or SR protocols is the degree of replication *drep* and the relative page access frequency $\frac{f_s}{f_c}$. There are interwoven aspects the SMM needs to consider:

1. Under what degree of replication the SR or CL protocol may be employed;
2. What the cost and QoS considerations are that influence a switch between SR and CL protocols.

These aspects are determined by the SP's goals on revenue accruals from its CDN service offerings. Figure 9 shows a profile of the protocol choices made at various points in time for a sample scenario of the changes in *drep* and client-level access patterns.

When a switch is made from the CL to the SR scheme (or vice versa), a hysteresis parameter is used in deciding when to switch back from the SR to the CL scheme. The hysteresis is to provide a 'control-theoretic' stability of the access protocol activities relative to the CDN operating conditions.

4.6 Simulation of update protocols

We have carried out a discrete-event simulation of CDN management protocols for the CL and SR schemes. The simulation involves first setting up a 'connectivity tree' over a subset of nodes in the physical topology of network that connects the content server, the server proxies, and the clients. The content update paths from the server to its proxies and the content access paths from the clients to their nearest proxies are set up over the tree. After these paths are set up, we simulated the CL and SR schemes in the various proxy nodes, on-tree content forwarding nodes, and server/client nodes.

The parameters of interest are the content access frequency by clients (f_c) and the content update fre-

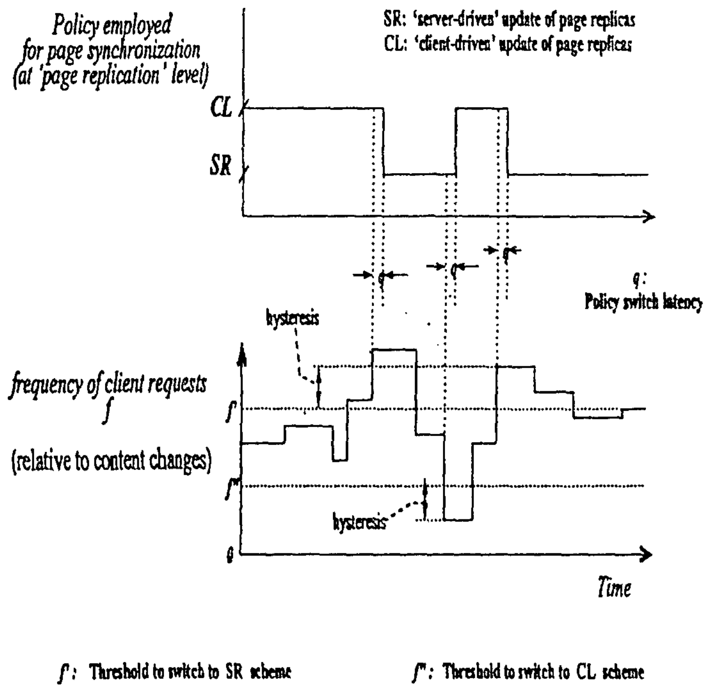

Policy employed for page synchronization (at 'page replication' level)
SR: 'server-driven' update of page replicas
CL: 'client-driven' update of page replicas
CL
SR
q: Policy switch latency
hysteresis
frequency of client requests f (relative to content changes)
f'
f''
0
hysteresis
Time
f': Threshold to switch to SR scheme
f'': Threshold to switch to CL scheme

Figure 9: Time-scales of parameters variations at different layers of a CDN

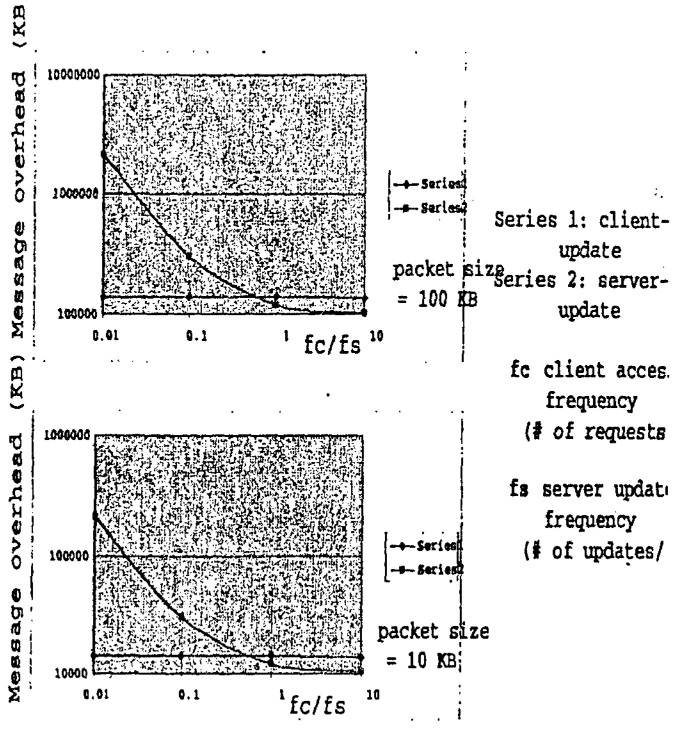


Figure 10: Simulation results on CDN protocols

quency by the server (f_s). Other control parameters used in simulation are the content page sizes, control message lengths, and proxy processing capacity. Simulation results on the message overhead incurred for content accesses were collected for various values of f_c, f_s, and page sizes. Figure 10 shows the simulation results when $\frac{f_c}{f_s}$ is varied from .01 to 10 (in logarithmic scales) and for page sizes of 10 *Kbytes* and 100 *Kbytes*. The results corroborate our earlier empirical observations on the relative costs incurred by the CL and SR schemes — c.f. Figure 6.

A suitable execution model of the SMM is required to allow management-oriented interactions with a target protocol system — such as CDNs. Here, the latter may consist of processes running on the various functional units involved in a service offering. The SMM processes need to signal the target protocol processes to enable their interactions. Details of the execution model are outside the scope of our paper.

5 Existing network service models

The OSI management model [9] deals with monitoring a restricted set of packet-level parameters at the transport and network layers. Examples are the number of packets of a particular classification flowing through a router and the packet loss rate on a transport connection. Whereas, our management paradigm allows the monitoring of protocol parameters that are reflected onto from the transport and session layer resources during a service offering. With service-oriented monitoring, a wider range of transport and session layer resource parameters can be captured.

SNMP provides a sniffer tool to examine the packets exchanged through a protocol [10]. One useful SNMP feature is a prescription of packet filters based on logical expressions that select which packets are captured for further analysis (e.g., packets from an IP host to a specific ethernet interface). Possibly, the management station may extract a resource usage profile from the packet-level observations, using 'applicative' functions loaded by the SP (e.g., number of packets of a connection C observed over a time interval indicates the bandwidth usage by C).

The OSI and SNMP models do not embody the mechanisms to reason about infrastructure resource usages and external environment conditions from the packet-level observations. In contrast, our model embodies this service-level management functionality by distributed monitoring of the protocol state — which, in part, subsumes packet-level observations.

There has been a number of works in the direction of policy-driven network management, with specific focus on adaptive QOS support. For instance, [11] describes dynamic policy mechanisms for characterization of network resources (such as link bandwidth and end-station CPU load). How these mechanisms can be extended to cover a broader class of abstract resources in our context of service-level management is not clear. Likewise, [12] describes a resource query system for network aware applications. However, this work also centers more on managing only the basic types of network resources.

The goal of our approach, in contrast, is to manage network services — rather than networks themselves. Service-level management requires the monitoring of network resource usage and external environment parameters, based on rules that prescribe the protocol-level access to resources. Low level network parameters (such as packet statistics through a router) may however be used in deriving this resource-oriented view. Dynamic protocol plug-in is a specific manifestation of such a service-level management.

6 Conclusions

The paper described a *meta-level* management tool that reconfigures the service provisioning mechanisms in response to the changes in network infrastructure resources and service operating environment. The goal is to achieve cost optimality (in a performance sense) by resorting to *dynamic protocol plug-ins*.

To integrate such a management tool into application development environments, we premise that network services be offered through abstract interfaces, with a service provider (SP) maintaining multiple protocol modules to support its clientele. Each of the protocols is geared to operate under certain network conditions, with the SP selecting the right protocol to provide the service under the prevailing network conditions. Our model allows dynamic switching from one protocol module to another, based on a notion of cost associated with the infrastructure resource usage by protocols under various operating conditions.

The paper described the service-level management mechanisms (SMM) to realize the switching from one protocol to another for service offering at various client-prescribed quality levels. These mechanisms are based on our studies of different network services: CDNs and end-to-end 'data connectivity' control.

The SMM monitoring of external environment conditions and service-level QoS are domain-specific, interfacing with the infrastructure layer and protocol service layer through customized management APIs. The meta-level architecture of SMM is itself generic, usable across different SPs. This generality reduces the development costs of distributed networking software that otherwise ensue with rigid domain-specific implementations of adaptive network services.

References

[1] M. Subramanian. **Telecommunications Management Network.** Chap.11, *Network Management: Principles and Practice*, A-W Publ., 2000.

[2] Object Management Group. **The Common Object Request Broker: Architecture and Specification.** Rev.2.0, OMG, 1995.

[3] Y. Chen, R. Katz, and J. Kubiatowicz. **Dynamic Replica Placement for Scalable Content Delivery.** In proc. *First Intl. Workshop on Peer-to-Peer Systems*, LNCS-2429, Springer-Verlag, pp.306-318, 2002.

[4] M. Schwartz. In **Data Link Protocols: Examples and Performance Analysis.** Chapter, *Telecommunication Networks*, P-H Publ., 1990.

[5] K. Ravindran. **Management Interface for Programmable End-to-End 'data connectivity' Provisioning.** In *IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*, (E2EMON-IM2005), Nice (France), 2005.

[6] S. Keshav. **Scheduling.** In Chap. 9, *An Engineering Approach to Computer Networking*, Addison-Wesley Publ. Co., pp.209-260, 1996.

[7] H. M. Mason and H. R. Varian. **Pricing Congestible Network Resources.** In *IEEE Journal on Selected Areas in Communications*, vol.13, no.7, pp.1141-1149, Sept. 1995.

[8] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. van Steen. **Replication for Web Hosting Systems.** In *ACM Computing Surveys*, 36(3), Sept.2004.

[9] A. S. Tanenbaum. In **SNMP — Simple Network Management Protocol.** Chap. 7, *Computer Networks*, P-H publ., pp.630-643, 1996.

[10] M. Subramanian. **SNMP Management RMON.** Chap. 8, *Network Management: Principles and Practice*, A-W publ., 2000.

[11] S. Howard, H. Lutfiyya, M. Katachabaw, and M. Bauer. **Supporting Dynamic Policy Change Using CORBA System Management Facilities.** In Proc. *Integrated Network Management*, IM'97, San Diego (CA), May 1997.

[12] T. Gross, P. Steenkiste. and J. Subhlok. **Adaptive Distributed Applications on Heterogeneous Networks.** In Proc. *8th Heterogeneous Computing Workshop*, HCW'99, 1999.